Temporal Logics & Model Checking

Farn Wang
Dept. of Electrical Engineering
National Taiwan University

Specifications, descriptions, & verification

- specification:
  - The user's requirement
- description (implementation):
  - The user's description of the systems
  - No strict line between description and specification.
- verification:
  - Does the description satisfy the specification ?

Formal specification & automated verification

- formal specificaton:
  - specification with rigorous mathematical notations
- automated verification:
  - verification with support from computer tools.

Why formal specifications ?

- to make the engineers/users understand the system to design through rigorous mathematical notations.
- to avoid ambiguity/confusion/misunderstanding in communication/discussion/reading.
- to specify the system precisely.
- to generate mathematical models for automated analysis.
- *But according to Goedel's incompleteness theorem, it is impossible to come up with a complete specification.*

Why automated verification ?

- to somehow be able to verify complexer & larger systems
- to liberate human from the labor-intensive verification tasks
  - to set free the creativity of human
- to avoid the huge cost of fixing early bugs in late cycles.
- to compete with the core verification technology of the future.

Specification & Verification ?

- Specification → Complete & sound.
- Verfication
  - → Reducing bugs in a system.
  - → Making sure there are very few bugs.

Very difficult!

Competitiveness of high-tech industry!

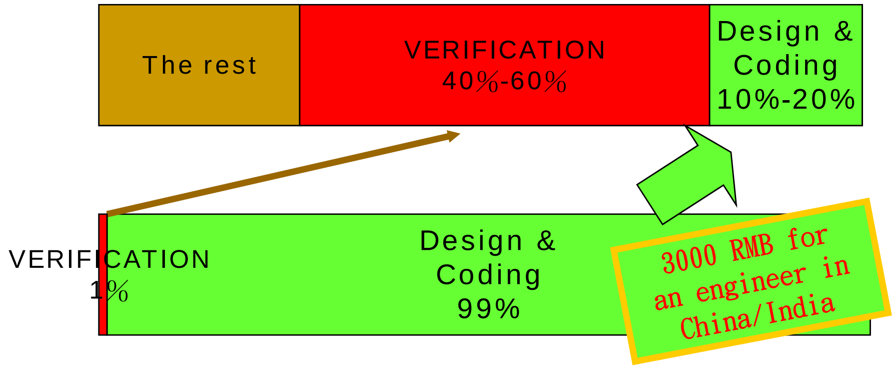
A way to survive for the students!

A way to survive for Taiwan!

Bugs in complex software

- They take effects only with special event sequences.
  - the number of event sequences is factorial and super astronomical!
- It is impossible to check all traces with test/simulation.

Budget appropriation

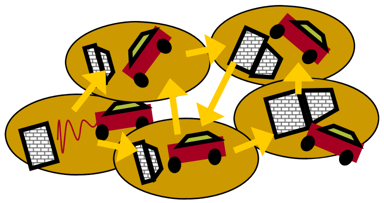


Training in Taiwan College

Three technologies in verification

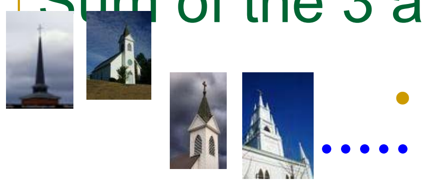
- Testing (real wall for real cars)
  - Expensive
  - Low coverage
  - Late in development cycles

- Simulation(virtual wall for virtual c
  - Economic
  - Low coverage
  - Don't know what you haven't seen.

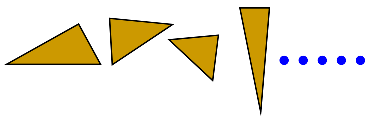


- Formal Verification (virtual car checked)
  - Expensive
  - Functional completeness
    - 100% coverage
  - Automated!
    - With algorithms and proofs.

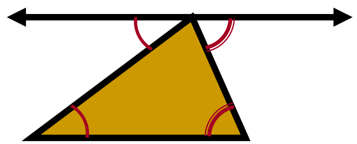
Sum of the 3 angles = 180 ?



- Testing (check all Δs you see)
  - Expensive
  - Low coverage
  - Late in development cycles

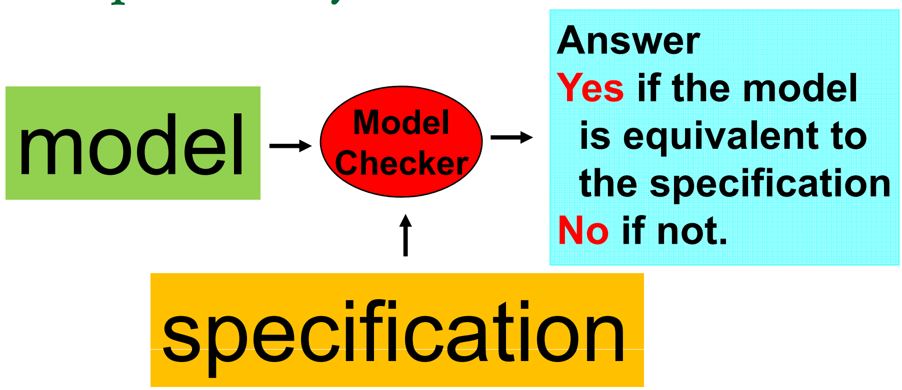


- Simulation (check all Δs you draw)
  - Economic
  - Low coverage
  - Don't know what you haven't seen.



- Formal Verification(we prove it.)
  - Expensive
  - Functional completeness
    - 100% coverage
  - Automated!
    - With algorithms and proofs.

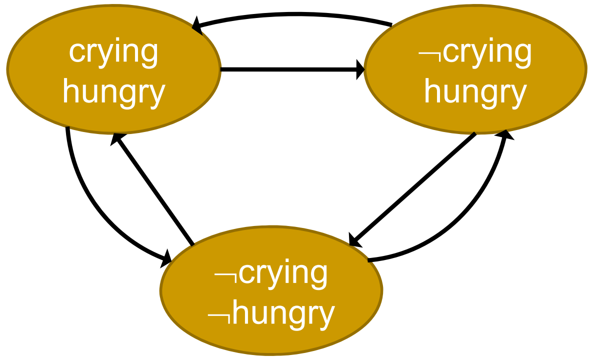
Model-checking
- a general framework for verification of sequential systems



Models & Specifications
- formalism

Whenever a baby cries, it is hungry.

- Logics: □(crying → hungry)
- Graphs:



Models & Specifications
- fairness assumptions

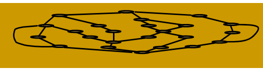
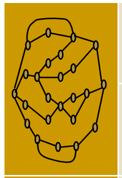
Some properties are almost impossible to verify without assumptions.

Example: □(start → ◇ finish)

To verify that a program halts, we assume

- CPU does not burn out.
- OS gives the program a *fair* share of CPU time.
- All the drivers do not stuck.
- ………

Model-checking
- frameworks in our lecture

Model \ Spec							Logics			
			traces		Trees		Linear		Branching	
			F=∅	F≠∅	F=∅	F≠∅	F=∅	F≠∅	F=∅	F≠∅
	traces	F=∅	✓	✓			✓	✓		
		F≠∅	✓	✓			✓	✓		
	Trees	F=∅			☑	✓			☑	✓
		F≠∅			✓	✓			✓	✓
Logics	Linear	F=∅					☑	☑		
		F≠∅					☑	☑		
	Branching	F=∅							✓	✓
		F≠∅							✓	✓

✓: known; ☑: discussed in the lecture

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

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)
- Branching
  - CTL (Computation Tree Logics)
  - CTL* (the full branching temporal logics)

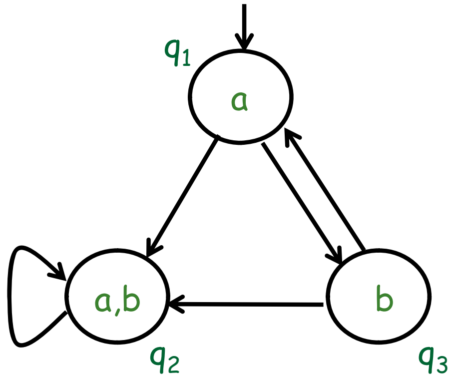
Kripke structure

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

- 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
- $L : S \mapsto 2^P$
  - 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

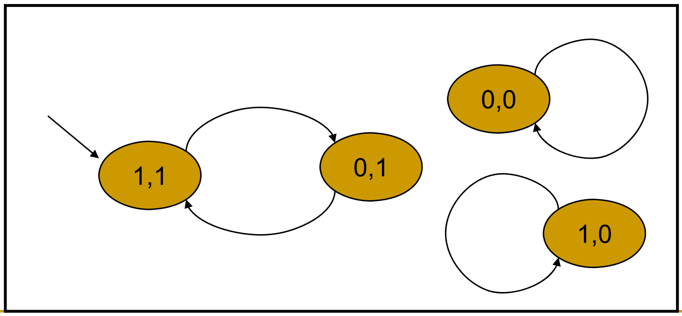
Suppose there is a program

```
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*x*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}



BNF, syntax definitions Note!

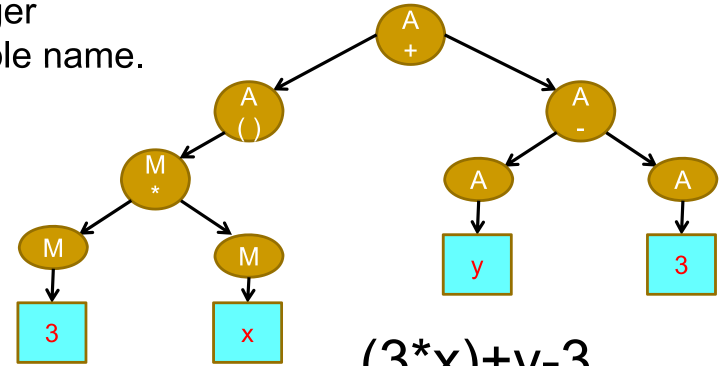
Be sure how to read BNF !

- used for define syntax of context-free language
- important for the definition of
  - automata predicates and
  - temporal logics
- Used throughout the lectures!
- In exam: violate the syntax rules ➔ no credit.

A ::= c | x | (M) | A_1+A_2 | A_1–A_2
M ::= c | x | (A) | M_1*M_2 | M_1/M_2
c is an integer
x is a variable name.

BNF, syntax definitions

A ::= c | x | (M) | A_1+A_2 | $A_1–A_2$
M ::= c | x | (A) | M_1*M_2 | M_1/M_2
c is an integer
x is a variable name.

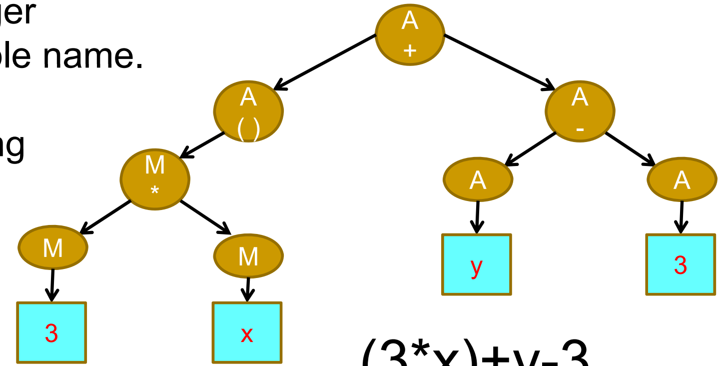


(3*x)+y-3

BNF, syntax definitions
- derivation trees (from top down)

A ::= c | x | (M) | A_1+A_2 | $A_1–A_2$
M ::= c | x | (A) | M_1*M_2 | M_1/M_2
c is an integer
x is a variable name.

used in string generation.

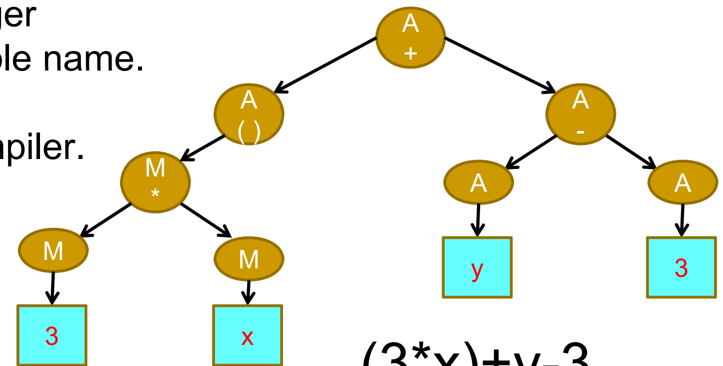


(3*x)+y-3

BNF, syntax definitions
- parsing trees (from bottom up)

A ::= c | x | (M) | A_1+A_2 | $A_1–A_2$
M ::= c | x | (A) | M_1*M_2 | M_1/M_2
c is an integer
x is a variable name.

used in compiler.



(3*x)+y-3

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)
    - LTL, PTL, PLTL
- Branching
  - CTL (Computation Tree Logics)
  - CTL* (the full branching temporal logics)

Amir Pnueli
1941

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

Presentation of a gift at ATVA /FORTE 2005, Taipei

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 ***P*** ***{true,false}***

or $s_k \in 2^P$

example: P={a,b}

{a}{a,b}{a}{a}{b}…

Syntax definitions Note!

Be sure how to read BNF !

- used for define syntax of context-free language
- important for the definition of
  - automata predicates and
  - temporal logics
- Used throughout the lectures!
- In exam: violate the syntax rules ➔ **no credit.**

A ::= (M) | A1 + A2 | A1 – A2
M ::= (A) | M1 * M2 | M1 / M2

LPTL
- syntax

syntax definition in BNF

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

abbreviation

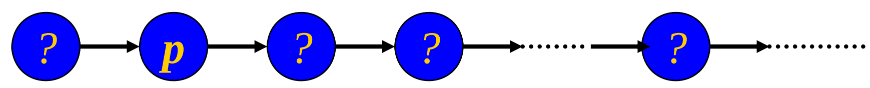
$$\mathbf{false} \equiv \neg\,\mathrm{true}$$
$$\psi_1 \wedge \psi_2 \equiv \neg((\neg\psi_1) \vee (\neg\psi_2))$$
$$\psi_1 \rightarrow \psi_2 \equiv (\neg\psi_1) \vee \psi_2$$
$$\Diamond\psi \equiv \mathbf{true}\ \mathcal{U}\psi$$
$$\Box\psi \equiv \neg\Diamond\neg\psi$$

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

$\bigcirc p$ $\mathbf{X}p$ ***p is*** **true on next state**

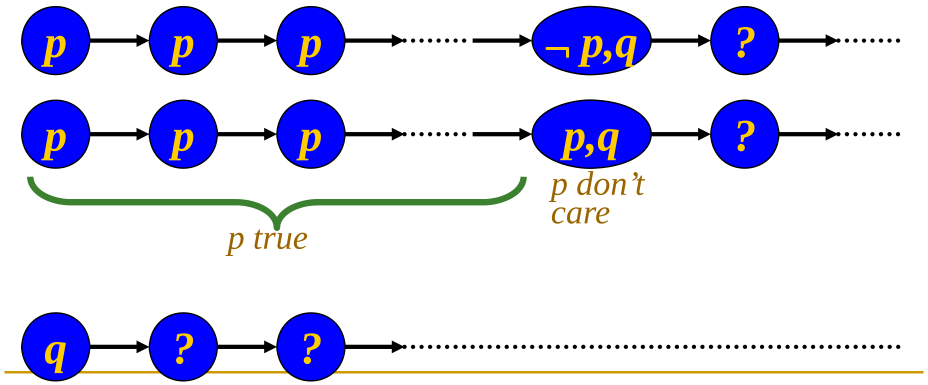


? : don’t care

LPTL

- syntax

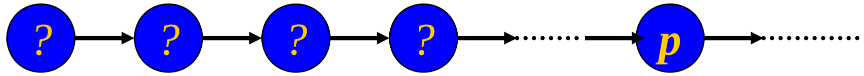
$p \cup q$ | **$p \cup q$** | **From now on, *p* is always true until *q* is true**



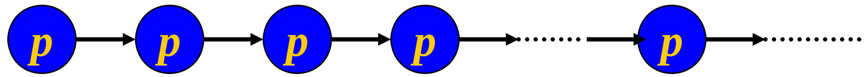
LPTL

- 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 ψ) inductively as :

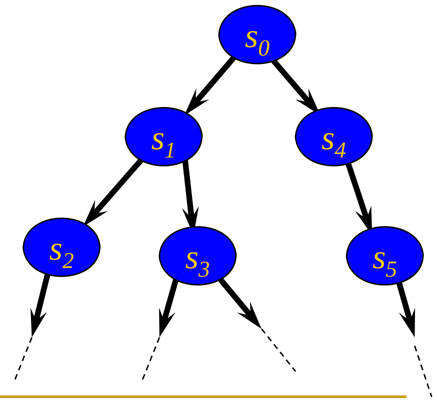
- $S \models$ true
- $S \models p \Leftrightarrow s_0(p)$=true, or equivalently $p \in s_0$
- $S \models \neg\psi \Leftrightarrow S \models \psi$ is false
- $S \models \psi_1 \vee \psi_2 \Leftrightarrow S \models \psi_1$ or $S \models \psi_2$
- $S \models \bigcirc\psi \Leftrightarrow S^{(1)} \models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

Branching Temporal Logics

Basic assumption of tree-like structure

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

•Every state may have many successors

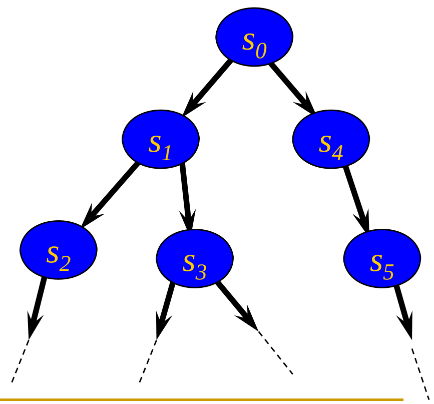


Branching Temporal Logics

Basic assumption of tree-like structure

•Every path is isomorphic as ***N***

- •Correspond to a state sequence



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

$s_0\ s_1\ s_2 \ldots\ldots$

$s_1\ s_3 \ldots\ldots$

$s_4\ s_5 \ldots\ldots$

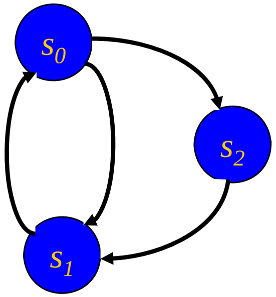
Branching Temporal Logic

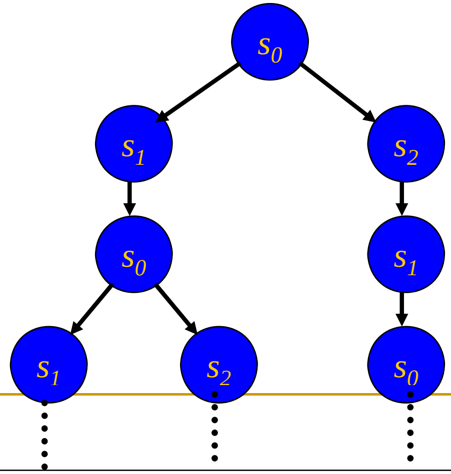
It can accommodate infinite and dense state successors

- 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

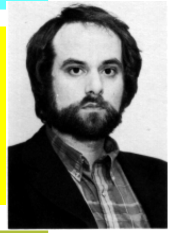




CTL(Computation Tree Logic)

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

φ::= true | p | $\neg\varphi$ | $\varphi_1 \vee \varphi_2$ | $\exists\bigcirc\varphi$ | $\forall\bigcirc\varphi$
| $\exists\varphi_1 U \varphi_2$ | $\forall\varphi_1 U \varphi_2$

abbreviation：

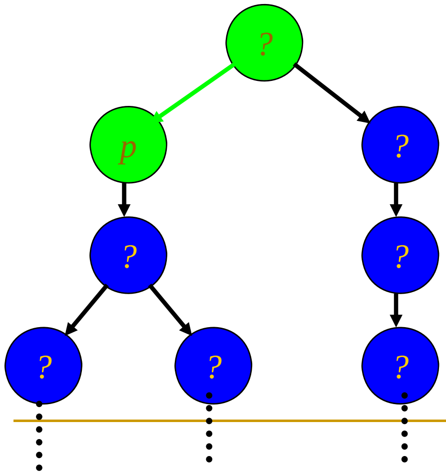
false	$\equiv$	$\neg$ true
$\varphi_1 \wedge \varphi_2$	$\equiv$	$\neg((\neg\varphi_1) \vee (\neg\varphi_2))$
$\varphi_1 \rightarrow \varphi_2$	$\equiv$	$(\neg\varphi_1) \vee \varphi_2$
$\exists\Diamond\varphi$	$\equiv$	$\exists$true $U\varphi$
$\forall\Box\varphi$	$\equiv$	$\neg\exists\Diamond\neg\varphi$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

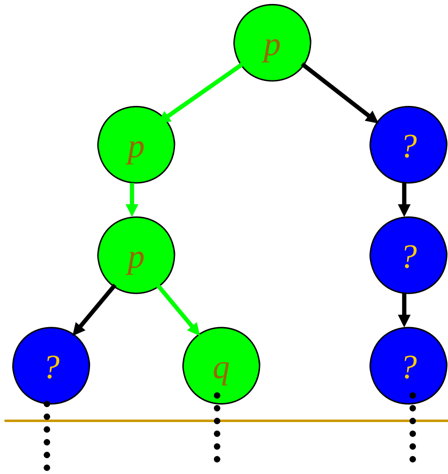
CTL
- semantics

$\exists \bigcirc p$ **EX*p*** **there exists a path where *p* is true on next state**



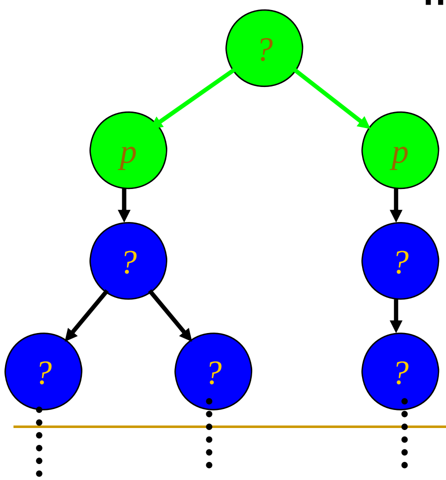
CTL
- semantics

$\exists p \, \mathrm{U} \, q$ ***p*EU*q*** **from now on, there is a path where *p* is always true until *q* is true**



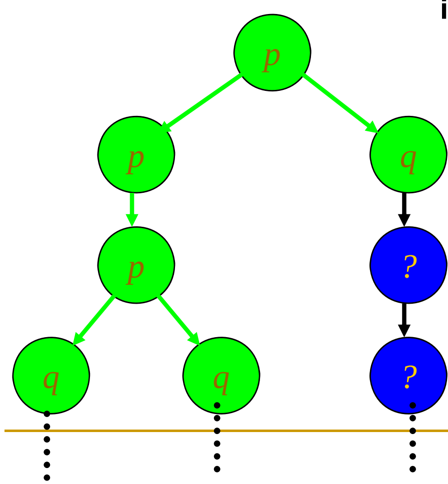
CTL
- semantics

$\forall \bigcirc p$ **AX*p*** **for all path where *p* is true on next state**



CTL
- semantics

$\forall p \, \mathrm{U} \, q$ ***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

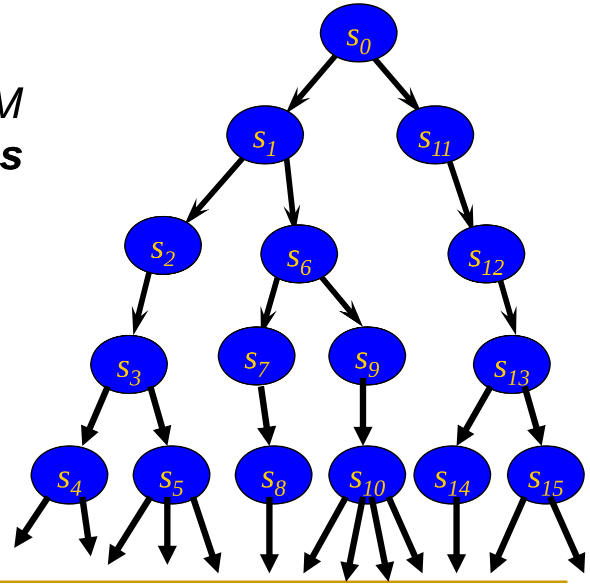
s-path : a path in *M* which stats from ***s***

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

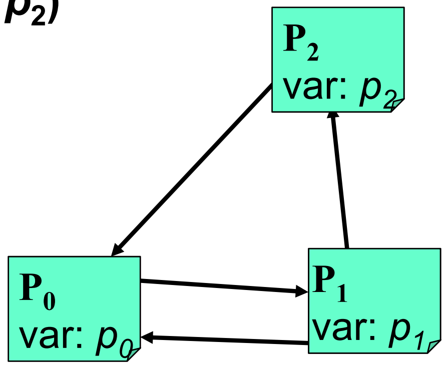
- M,s ⊨ true
- M,s ⊨ p ⇔ p ∈s
- M,s ⊨ ¬φ ⇔ it is false that M,s ⊨ φ
- M,s ⊨ $\varphi_1 \vee \varphi_2$ ⇔ M,s ⊨ φ_1 or M,s ⊨ φ_2
- M,s ⊨ ∃○φ ⇔ ∃ s-path = $s_0\, s_1$ ......(M,s_1 ⊨ φ)
- M,s ⊨ ∀○φ ⇔ ∀ s-path = $s_0\, s_1$ ......(M,s_1 ⊨ φ)
- M,s ⊨ $\exists \varphi_1 U \varphi_2$ ⇔ ∃ s-path = $s_0\, s_1$ ......, ∃k≥0 (M,s_k ⊨ $\varphi_2 \wedge \forall 0 \leq j < k$(M,$s_j$ ⊨ φ_1)
- M,s ⊨ $\forall \varphi_1 U \varphi_2$ ⇔ ∀s-path = $s_0\, s_1$ ......, ∃k≥0 (M,s_k ⊨ $\varphi_2 \wedge \forall 0 \leq j < k$(M,$s_j$ ⊨ φ_1)

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.

   ∀□(dark-clouds→ ∀◇rain)

2. if a buttefly flaps its wings, the New York stock could plunder.

   ∀□(buttefly-flap-wings →∃◇NY-stock-plunder)

3. if I win the lottery, I will be happy forever.

   ∀□(win-lottery →∀□ happy)

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

   ∀□(exec→∀○(intrpt→∀○(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.

∀□(exec→∀○(intrpt→∀○(intrpt-handler)))

Some possible mistakes:

∀□(exec→((∀○ intrpt)→∀○intrpt-handler))

∀□(exec → ((∀○ intrpt) →∀○∀○ intrpt-handler))

CTL
- examples (IIIa)

Please draw a Kripke structure that tells

∀○(intrpt→∀○(intrpt-handler))

from

(∀○ intrpt)→∀○intrpt-handler

and

(∀○ intrpt) →∀○∀○ intrpt-handler

CTL
- important classes

- ∀□η : safety properties
  - η is always true in all computations from now.
- ∃◇η: reachability properties
  - η is eventually true in some computation from now.
  - ∀□η ≡ ¬∃◇¬η
- ∀◇η: inevitabilities
  - η is eventually true in all computations from now.
- ∃□η
  - ∀◇η ≡ ¬∃□¬η

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

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

CTL* is the set of all state-fomulas!

CTL*
- examples (1/4)

In a fair concurrent environment, jobs will eventually finish.

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

or

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

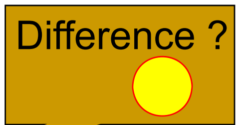
CTL*
- examples (2/4)

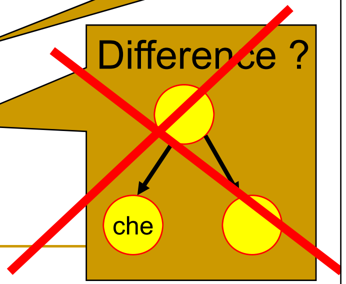
No matter what, infinitely many comets will hit earth.

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

Why not CTL?

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





Exercise, please construct a model that tells the last from the first

CTL*
- examples (2/4)

No matter what, infinitely many comets will hit earth.

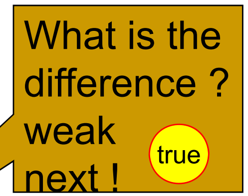
$\forall\Box\Diamond$ comet-hit-earth

Or

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

Why not CTL?

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



CTL*
- Workout

The same according to lemma

- **(1)** $\forall\Box\Diamond$comet-hit-earth
- **(2)** $\forall\Box\ \forall\ \Diamond$ comet-hit-earth
- **(3)** $\forall\Box\ \exists\ \Diamond$ comet-hit-earth

Please draw Kripke structures 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.

$\forall((\Box$you-have-no-lover$)\rightarrow\Diamond$marry-you$)$

Why not CTL ?

- $(\forall\Box$ you-have-no-lover$)\rightarrow\forall\ \Diamond$ marry-you
- $(\forall\Box$ you-have-no-lover$)\rightarrow\exists\ \Diamond$ marry-you
- $(\exists\Box$ you-have-no-lover$)\rightarrow\forall\ \Diamond$ marry-you

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

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 :

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

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

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

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

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

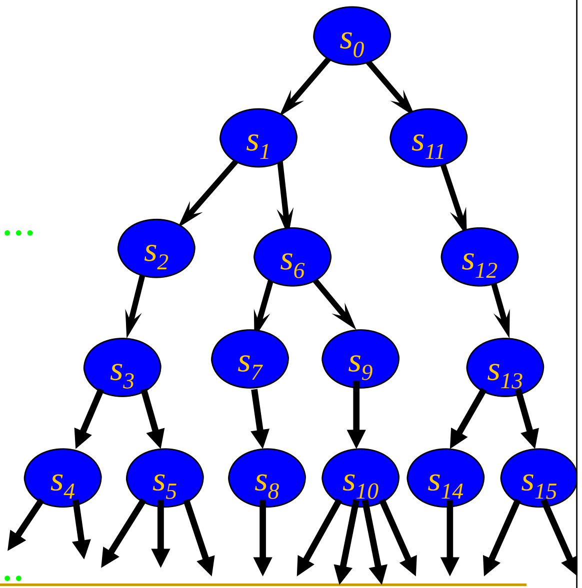
$S^{(4)} = s_5 \ldots\ldots$

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

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

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

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



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

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

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

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 $\forall \varphi \in L_2$ **([φ] is expressible in 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
- 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

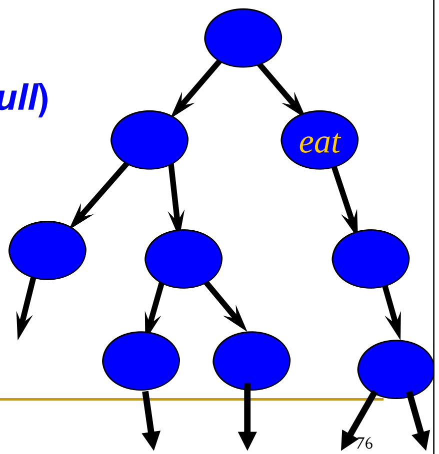
Very little known at the moment,

the fine difference in semantics of branching-structures

Expressiveness
- CTL*, example (I)

A tree the distinguishes the following two formulas.

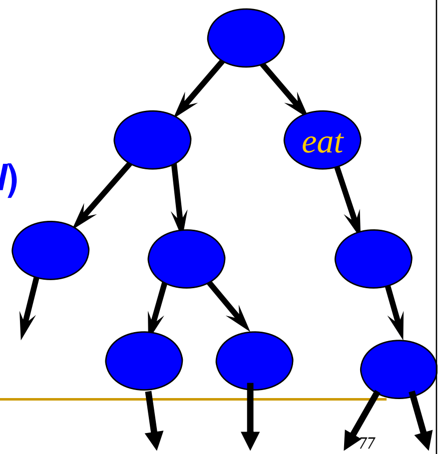
- $\forall((\Diamond eat) \rightarrow \Diamond full)$
  - **Negation:** $\exists((\Diamond eat) \wedge \Box \neg full)$
- $(\forall \Diamond eat) \rightarrow (\forall \Diamond full)$



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 $\exists$, $\forall$, we have many CTL* subclasses.

Example:

$B(\neg,\vee,\bigcirc,U)$: only $\neg,\vee,\bigcirc,U$ *after* $\exists$, $\forall$

$B(\neg,\vee,\bigcirc,\Diamond^{\infty})$: only $\neg,\vee,\bigcirc,\Diamond^{\infty}$ after $\exists$, $\forall$

$B(\bigcirc,\Diamond)$: only $\bigcirc,\Diamond$ after $\exists$, $\forall$

Expressiveness
- CTL*

CTL* subclass expressiveness heirarchy

CTL* $>$ $B(\neg,\vee,\bigcirc,\Diamond,U,\Diamond^{\infty})$

$>$ $B(\bigcirc,\Diamond,U,\Diamond^{\infty})$

$>$ $B(\neg,\vee,\bigcirc,\Diamond,U)$

$=$ $B(\bigcirc,\Diamond,U)$

$>$ $B(\neg,\vee,\bigcirc,\Diamond)$

$>$ $B(\bigcirc,\Diamond)$

$>$ $B(\Diamond)$

Expressiveness
- CTL*

Some theorems :

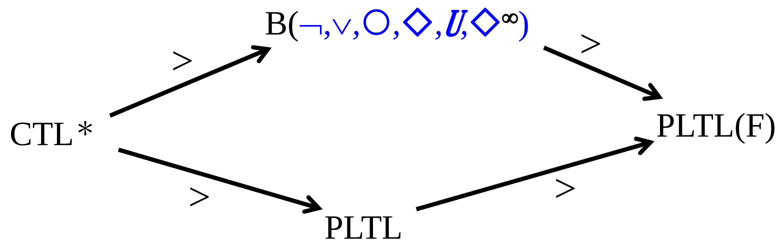
- $B(\neg,\vee,\bigcirc,\Diamond,U) \equiv B(\bigcirc,\Diamond,U)$
- **$\exists\Diamond^{\infty}p$ is inexpressible in $B(\bigcirc,\Diamond,U)$.**

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.



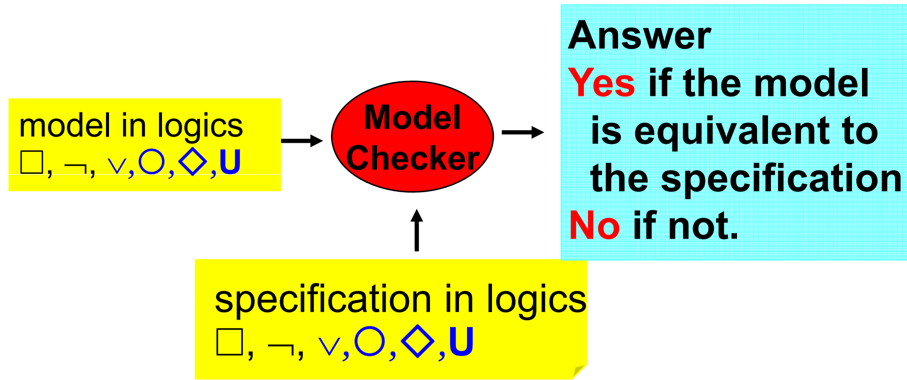
Verification

model (system) formula

specification formula

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

Satisfiability-checking framework



LPTL
- tableau for satisfiability checking

Tableau for φ

- a finite Kripke structure that fully describes the behaviors of φ
- 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(φ) (closure) is the smallest set of formulas containing φ 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 ψ_1 , ψ_2 , $\bigcirc (\psi_1 \mathbf{U} \psi_2) \in CL(\varphi)$
- If $\Box \psi \in CL(\varphi)$, then ψ, $\bigcirc \Box \psi \in CL(\varphi)$

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 $(v,v') \in E$, if $\bigcirc \psi \in v$, then $\psi \in v'$

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

LPTL
- tableau for satisfiability checking

CL(pUq) = {pUq, $\bigcirc$pUq, p, $\neg$ p, q, $\neg$ q }

Example: (p U q)

tableau (V,E)

V: {p, q, pUq , $\bigcirc$pUq} {p, q, $\bigcirc$pUq} {p, q}

{p, q, pUq}

{p, $\neg$q, pUq, $\bigcirc$pUq} {p, $\neg$q, $\bigcirc$pUq} {p, $\neg$q}

{$\neg$p, q, pUq, $\bigcirc$pUq} {$\neg$p, q, pUq} {$\neg$p,q}

{$\neg$p, q, $\bigcirc$pUq}

{$\neg$p, $\neg$q, $\bigcirc$pUq} {$\neg$p, $\neg$q}

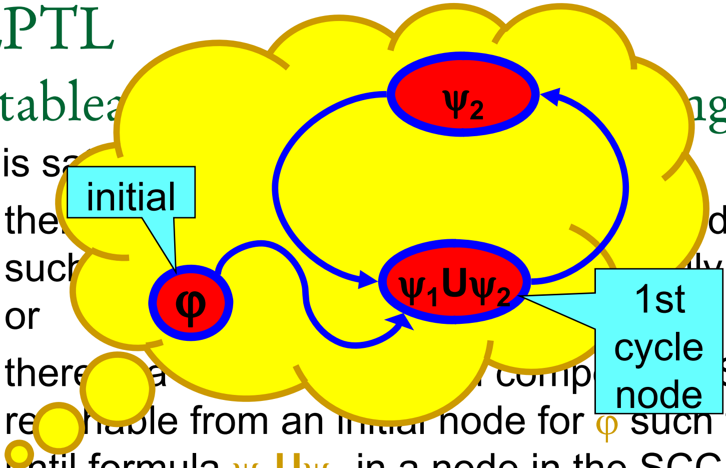
E: ?

LPTL

- tablea… …ng

φ is sa…

- the… …de for φ such… …ly satisfied; or
- there… a… …comp… …SCC) re…able from an … node for φ such that for all until formula $\psi_1 U \psi_2$ in a node in the SCC, there is also a node in the SCC containing ψ_2 ; or
- there is a cycle reachable from an initial node for φ such that the for all until formulas $\psi_1 U \psi_2$ in **the first cycle node**, there is also a node in the cycle containing ψ_2 .



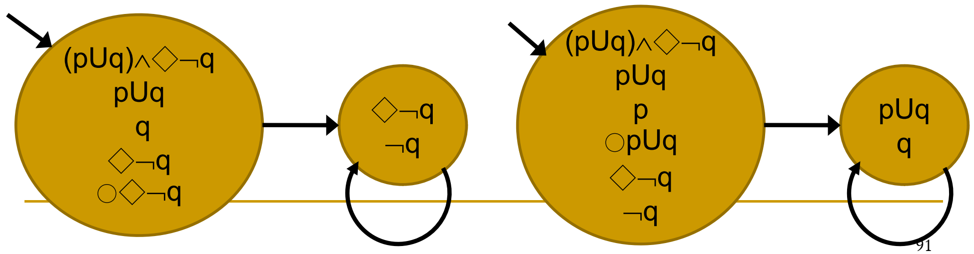
LPTL

- 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

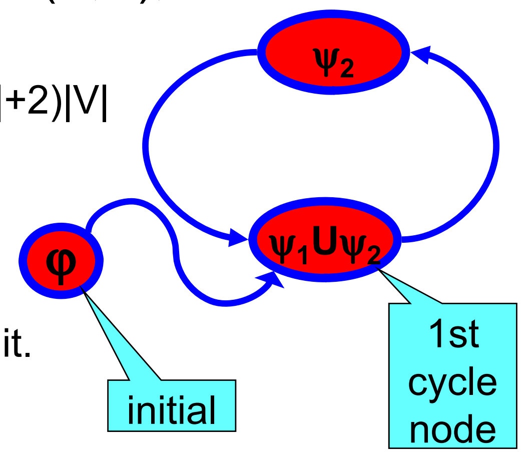
LPTL

- tableau for satisfiability checking

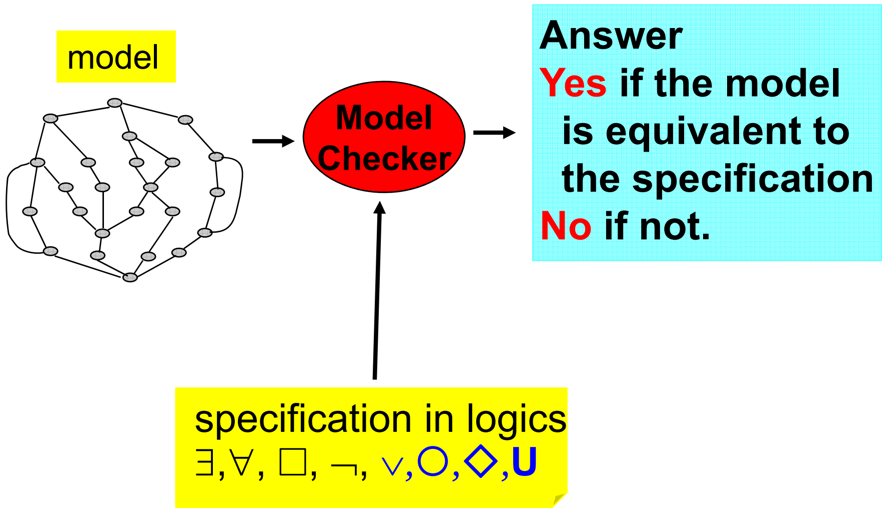
φ is satisfiable iff in (V,E),

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.



CTL model-checking framework



CTL
- model-checking

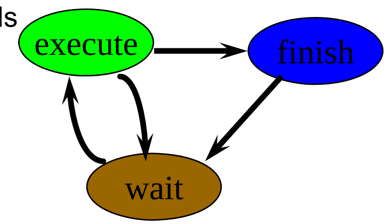
Given a finite Kripke structure M and a CTL formula φ, is M a model of φ ?

- 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.

Kripke structure
- *Least fixpoint in modal logics*

Dark-night murder, strategy I:

A suspect will be in the 2nd round iff

- *He/she lied to the police in the 1st round; or*
- *He/she is loyal to someone in the 2nd round*

What is the minimal solution to *2nd[]* ?

$$2nd[i] \equiv Liar[i] \vee \exists j{\neq}i(2nd[j] \wedge Loyal\text{-}to[i,j])$$

Kripke structure

- *Least fixpoint in modal logics*

In a dark night, there was a cruel murder.

- n suspects, numbered 0 through n-1.
- *Liar[i]* iff suspect i has lied to the police in the 1st round investigation.
- *Loyal-to[i,j]* iff suspect i is loyal to suspect j in the same criminal gang.
- *2nd[i]* iff suspect i to be in 2nd round investigation.

What is the minimal solution to *2nd[]* ?

Kripke structure

- *Greatest fixpoint in modal logics*

In a dark night, there was a cruel murder.

- n suspects, numbered 0 through n-1.
- *¬Liar[i]* iff the police cannot prove suspect i has lied to the police in the 1st round investigation.
- *Loyal-to[i,j]* iff suspect i is loyal to j are in the same criminal gang.
- *2nd[i]* iff suspect i to be in 2nd round investigation.

What is the maximal solution to *¬ 2nd[]* ?

Kripke structure

- *Greatest fixpoint in modal logics*

Dark-night murder, strategy II

A suspect will not be in the 2nd round iff

- *We cannot prove he/she has lied to the police; and*
- He/she is loyal to someone not in the 2nd round.

What is the maximal solution to *¬ 2nd[]* ?

$$\neg 2nd[i] \equiv \neg Liar[i] \wedge \exists j \neq i (\neg 2nd[j] \wedge Loyal\text{-}to[i,j])$$

In comparison:

$\neg 2nd[i] \equiv \neg Liar[i] \wedge \forall j \neq i (\neg 2nd[j] \wedge Loyal\text{-}to[i,j])$

$\neg 2nd[i] \equiv \neg Liar[i] \wedge \forall j \neq i (\neg 2nd[j] \rightarrow Loyal\text{-}to[i,j])$

$\neg 2nd[i] \equiv \neg Liar[i] \wedge \forall j \neq i (Loyal\text{-}to[i,j] \rightarrow \neg 2nd[j])$

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 ?

$$\forall \Box p \equiv \neg \exists \Diamond \neg p \equiv \neg \exists true\ U \neg p$$

Reachability analysis: $\exists\Diamond\eta$

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

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

Kripke structure - safety analysis

Reachability algorithm in graph theory

Given

- a Kripke structure $A = (S, S_0, R, L)$
- a safety predicate **η**,

find a path from a state in S_0 to a state in $[\neg\eta]$.

Solutions in graph theory

- Shortest distance algorithms
- spanning tree algorithms

Kripke structure - safety analysis

```
/* Given A = (S, S0, R, L)*/
safety_analysis(η) /* using least fixpoint algorithm */ {
   for all s, if ¬η∈L(s), L(s)=L(s)∪{∃◇¬η};
   repeat {
      for all s, if ∃(s,s')(∃◇¬η∈L(s')),
         L(s)=L(s)∪{∃◇¬η};
   } until no more changes to L(s) for any s.
   if there is an s0∈S0 with ∃◇¬η∈L(s0),
      return 'unsafe,'
      else return 'safe.'
}
```

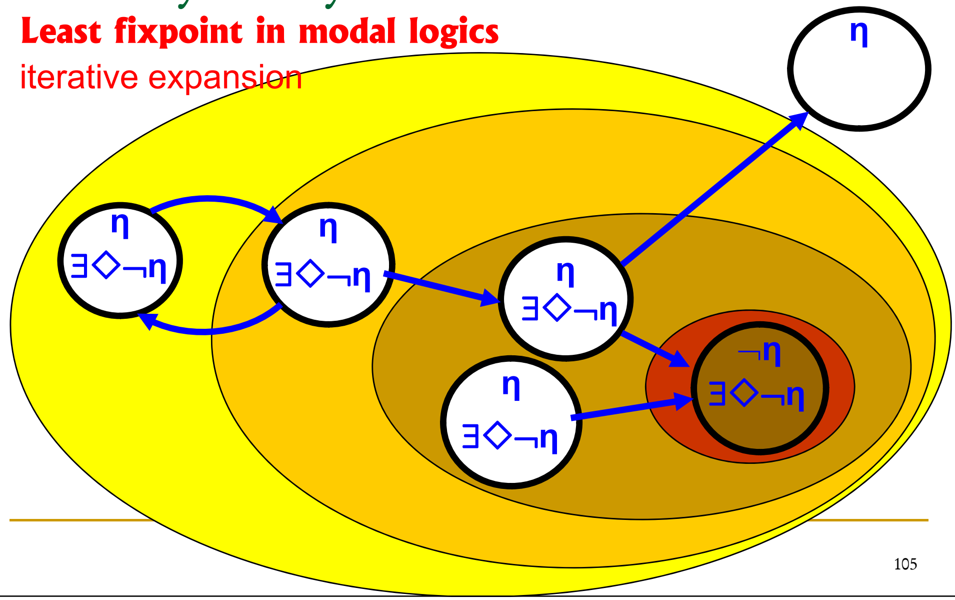
A notation for the possibility of $\neg\eta$

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

Kripke structure - safety analysis

Least fixpoint in modal logics

iterative expansion



Kripke structure
- liveness analysis : $\forall\Diamond\ \eta$

Given

- a Kripke structure $A = (S, S_0, R, L)$
- a liveness predicate η,

can η be true eventually ?

Example:

Can the computer be started successfully ?

Will the alarm sound in case of fire ?

Kripke structure
- liveness analysis

Strongly connected component algorithm in graph theory

Given

- a Kripke structure $A = (S, S_0, R, L)$
- a liveness predicate η,

find a cycle such that

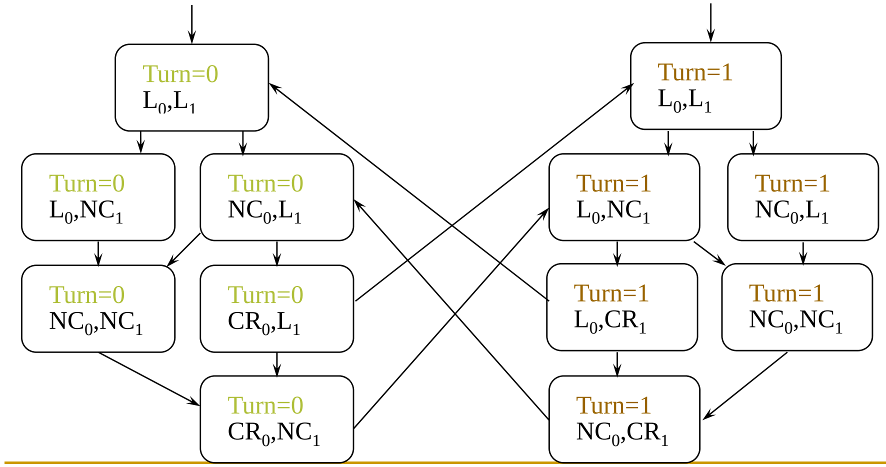
- all states in the cycle are $\neg\eta$
- there is a $\neg\eta$ path from a state in S_0 to the cycle.

Solutions in graph theory

- strongly connected components (SCC)

Kripke structure
- liveness analysis

$\Box$ (Turn=0 $\rightarrow\Diamond$ Turn=1)



Kripke structure
- liveness analysis

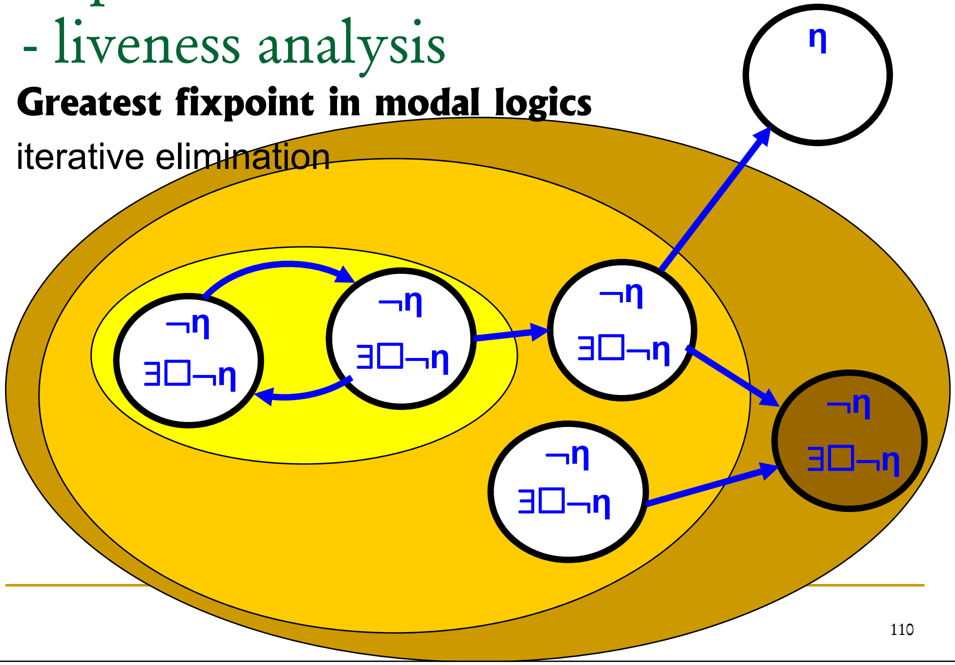
```
liveness(η) /* using greatest fixpoint algorithm */ {
   for all s, if ¬η∈L(s), L(s)=L(s)∪{∃□¬η};
   repeat {
      for all s, if ∃□¬η ∈L(s) and ∀(s,s')(∃□¬η ∉L(s)),
         L(s)=L(s) - {∃□¬η};
   } until no more changes to L(s) for any s.
   if there is an s0∈S0 with ∃□¬η∈L(s0),
      return 'liveness not true,'
      else return 'liveness true.'
}
```

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

Kripke structure
- liveness analysis

Greatest fixpoint in modal logics

iterative elimination



CTL model-checking

The NORMAL form needed in CTL model-checking:

1. only modal operators

$$\exists\bigcirc\varphi,\ \exists\ \psi_1 \mathbf{U} \psi_2,\ \exists\square\varphi$$

2. No modal operators

$$\forall\bigcirc\varphi,\ \forall\ \psi_1 \mathbf{U} \psi_2,\ \forall\square\varphi,\ \forall\Diamond\varphi,\ \exists\Diamond\varphi$$

3. No double negation: $\neg\neg\varphi$
4. No implication: $\psi_1 \Rightarrow \psi_2$

CTL
- model-checking algorithm (1/6)

Given M and φ,

1. Convert φ to NORMAL form.
2. list the elements in Cl (φ) according to their sizes

$$\varphi_0\ \varphi_1\ \varphi_2 \cdots \varphi_n$$

for all $0 \le i < j \le n$, φ_j is not a subformula of φ_i

2. for i=0 to n,

   label (φ_i)

3. for all initial states s_0 of M, if $\varphi \notin L(s_0)$, return `No!'
4. return `Yes!'

See next page!

CTL
- model-checking algorithm (2/6)

label(φ) {

case p, return;

case $\neg\varphi$, for all s, if $\varphi \notin L(s)$, $L(s) = L(s) \cup \{\neg\varphi\}$

case $\varphi\vee\psi$, for all s, if $\varphi \in L(s)$ or $\psi \in L(s)$,
  $L(s)=L(s)\cup\{\varphi\vee\psi\}$

case $\exists\bigcirc\varphi$, for all s, if $\exists(s,s')$ with $\varphi \in L(s')$,
  $L(s)=L(s)\cup\{\exists\bigcirc\varphi\}$

case $\exists\ \psi_1 \mathbf{U} \psi_2$, lfp($\psi_1$, ψ_2);

case $\exists\square\varphi$, gfp(φ);

}

CTL
- model-checking algorithm (3/6)

lfp(ψ_1 , ψ_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.

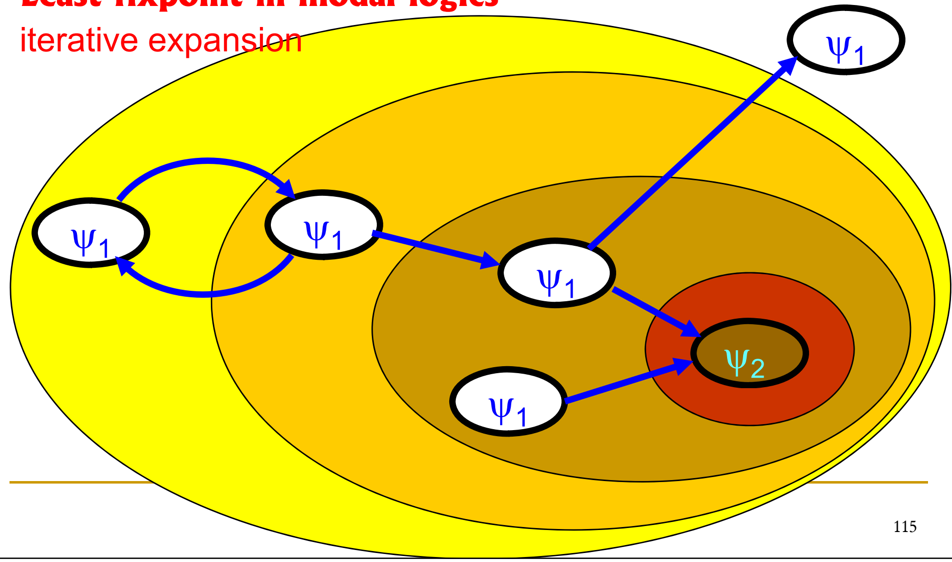
}

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.

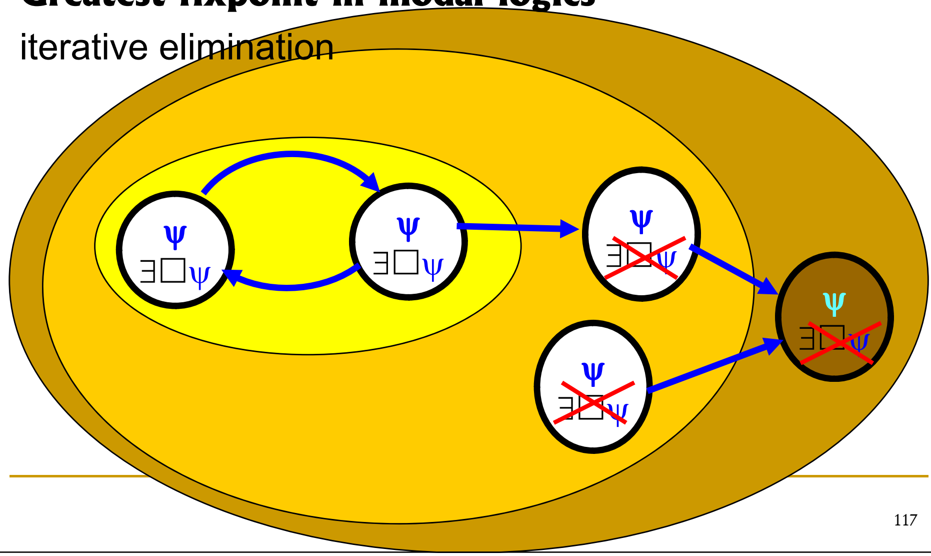
}

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

CTL
- model-checking algorithm (6/6)

Greatest fixpoint in modal logics

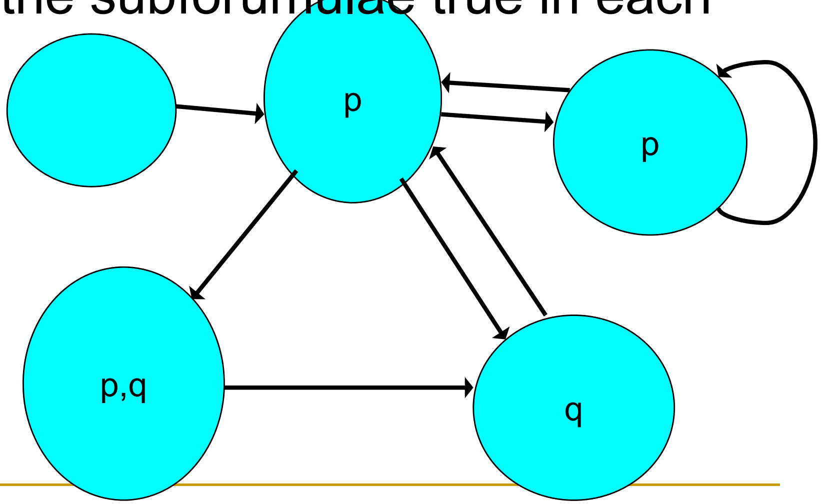
iterative elimination



(∃○∃p*U*q) ∧ ∃□p

Labeling funciton:

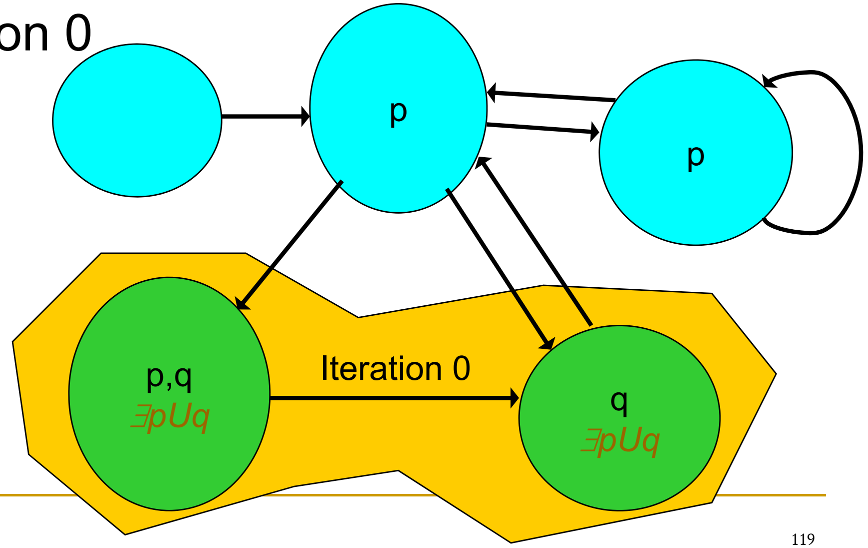
label the subforumulae true in each state.



(∃○∃p*U*q) ∧ ∃□p

Evaluating ∃pUq using least fixpoint

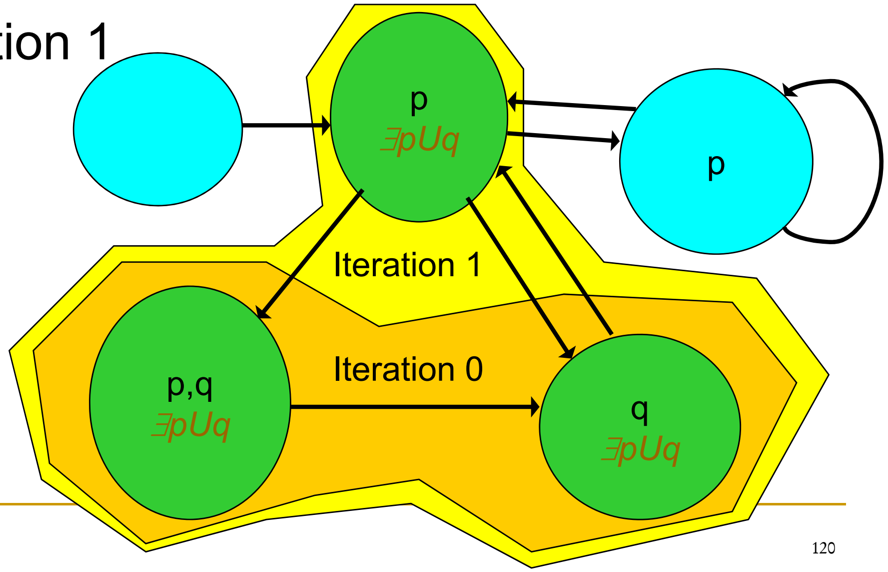
Iteration 0



(∃○∃p*U*q) ∧ ∃□p

Evaluating ∃pUq using least fixpoint

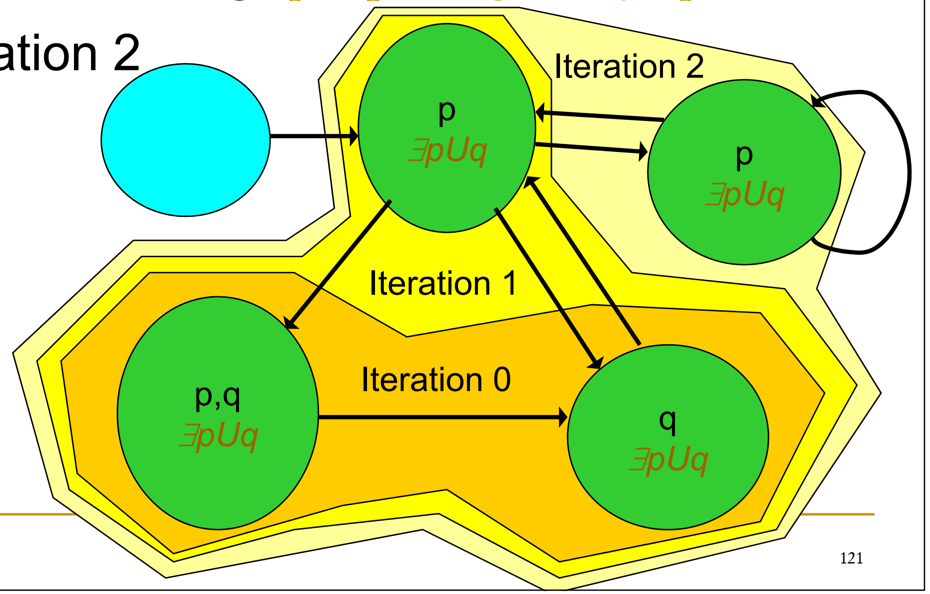
Iteration 1



(∃○∃p*U*q) ∧ ∃□p

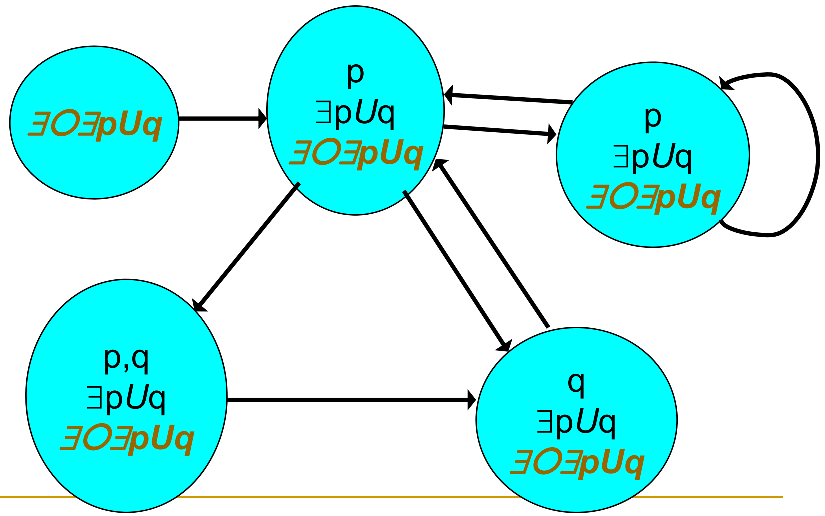
Evaluating ∃pUq using least fixpoint

Iteration 2



(∃○∃p*U*q) ∧ ∃□p

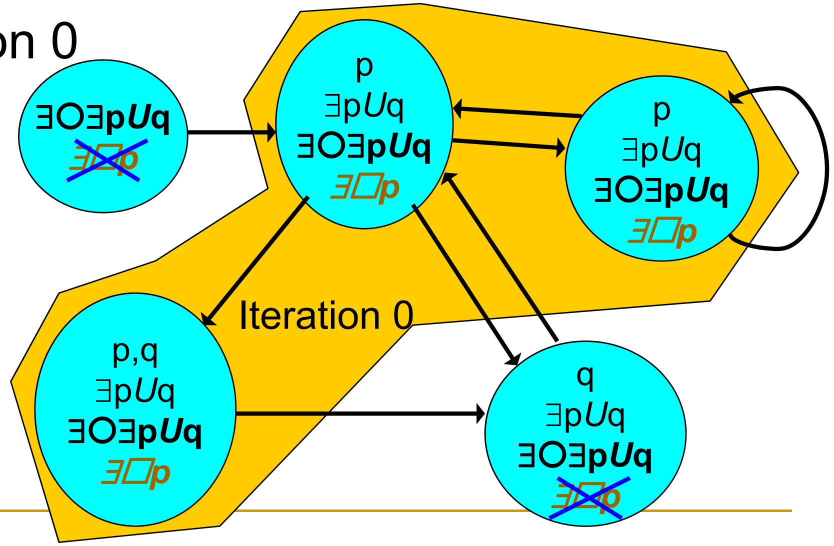
Evaluating ∃○∃pUq



(∃○∃p*U*q) ∧ ∃□p

Evaluating ∃□p using greatest fixpoint

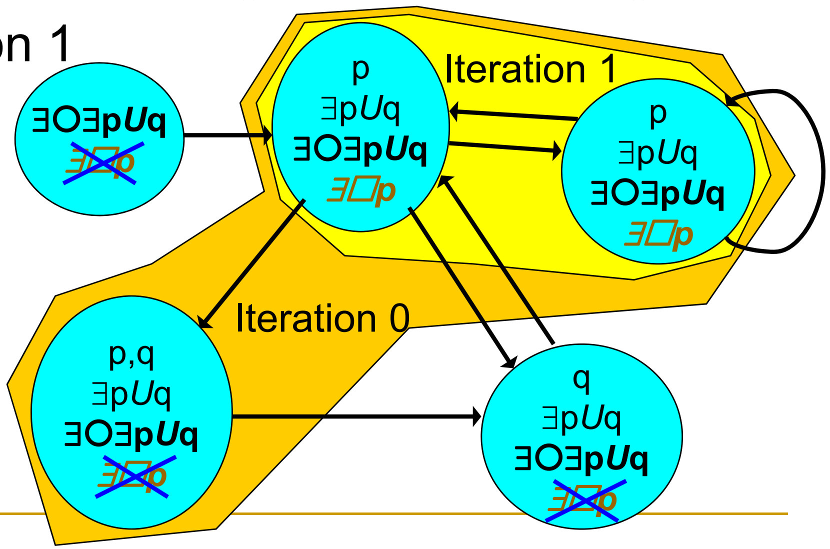
Iteration 0



(∃○∃p*U*q) ∧ ∃□p

Evaluating ∃□p using greatest fixpoint

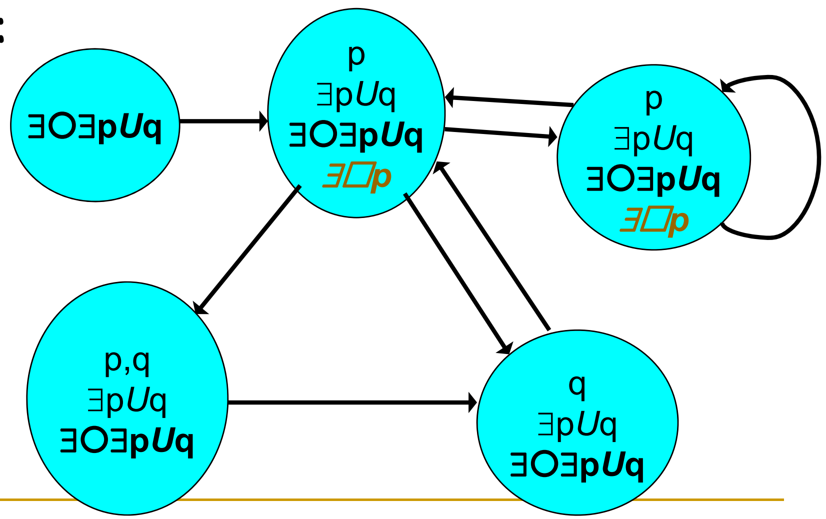
Iteration 1



(∃○∃p*U*q) ∧ ∃□p

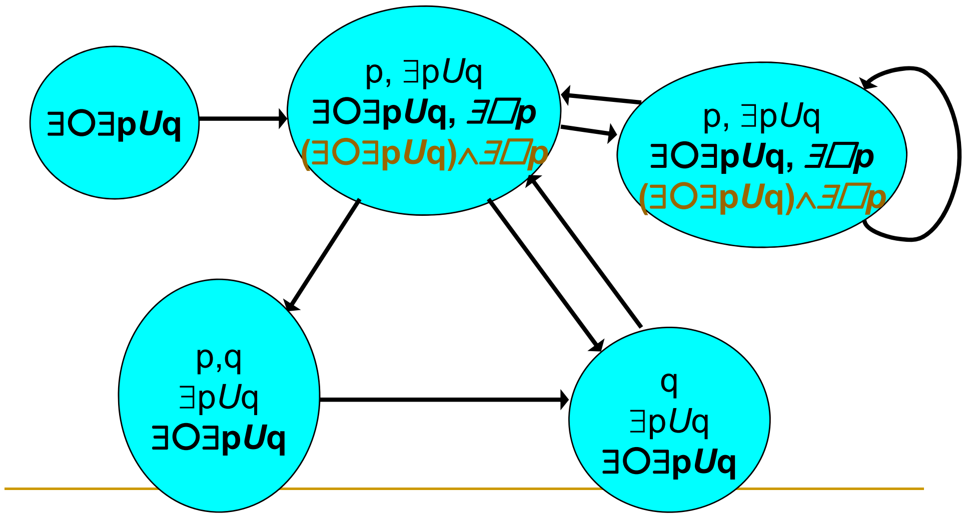
Evaluating ∃□p using greatest fixpoint

Result:

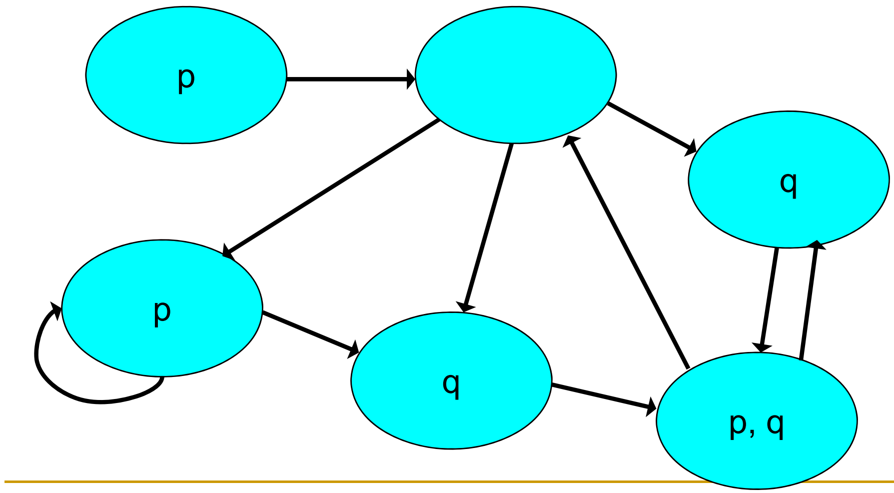


(∃○∃p*U*q) ∧ ∃□p

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.

CTL
- symbolic model-checking with BDD

- System states are described with binary variables.

  n binary variables → **2*ⁿ*** states

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

- we can use a BDD to describe legal states.

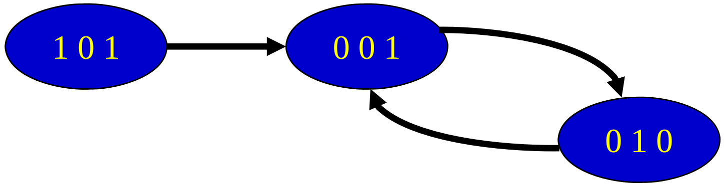
  a Boolean function with *n* binary variables

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

CTL - symbolic model-checking with Propositioal logics

Example:

$\boldsymbol{x_1 \;\; x_2 \;\; x_3}$

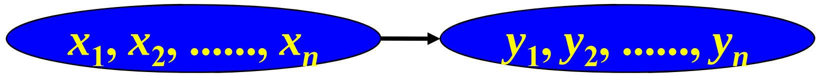


$$\text{state}(x_1, x_2, x_3) = (x_1 \wedge \neg x_2 \wedge x_3) \vee (\neg x_1 \wedge \neg x_2 \wedge x_3) \vee (\neg x_1 \wedge x_2 \wedge \neg x_3)$$

CTL - symbolic model-checking with Propositioal logics

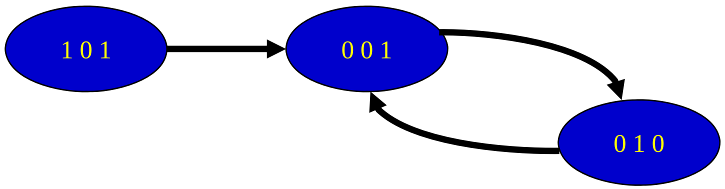
State transition relation as a logic funciton with <u>2*n* parameters</u>

$\text{transition}(x_1, x_2, \ldots\ldots, x_n, y_1, y_2, \ldots\ldots, y_n)$



CTL - symbolic model-checking with Propositioal logics

$x_1 \;\; x_2 \;\; x_3 \;\; y_1 \;\; y_2 \;\; y_3$

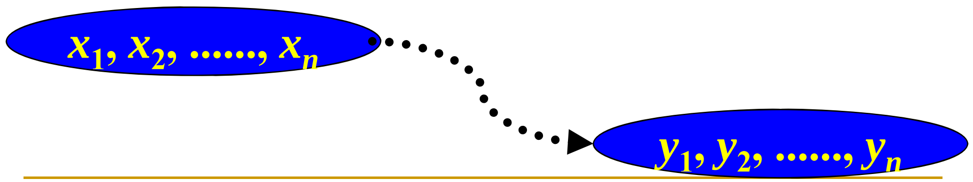


$$\text{transition}(x_1, x_2, x_3, y_1, y_2, y_3) = (x_1 \wedge \neg x_2 \wedge x_3 \wedge \neg y_1 \wedge \neg y_2 \wedge y_3) \vee (\neg x_1 \wedge \neg x_2 \wedge x_3 \wedge \neg y_1 \wedge y_2 \wedge \neg y_3) \vee (\neg x_1 \wedge x_2 \wedge \neg x_3 \wedge \neg y_1 \wedge \neg y_2 \wedge y_3)$$

CTL - symbolic model-checking with Propositioal logics

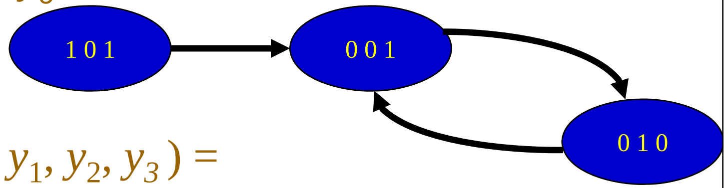
Path relation also as a logic funciton with <u>2*n* parameters</u>

$\text{reach}(x_1, x_2, \ldots\ldots, x_n, y_1, y_2, \ldots\ldots, y_n)$



CTL - symbolic model-checking with Propositial logics

$x_1 \ x_2 \ x_3 \ y_1 \ y_2 \ y_3$



$$reach(x_1, x_2, x_3, y_1, y_2, y_3) = (x_1\wedge\neg x_2\wedge x_3\wedge\neg y_1\wedge\neg y_2\wedge y_3) \vee (x_1\wedge\neg x_2\wedge x_3\wedge\neg y_1\wedge y_2\wedge\neg y_3) \vee (\neg x_1\wedge\neg x_2\wedge x_3\wedge\neg y_1\wedge y_2\wedge\neg y_3) \vee (\neg x_1\wedge x_2\wedge\neg x_3\wedge\neg y_1\wedge\neg y_2\wedge y_3) \vee (\neg x_1\wedge\neg x_2\wedge x_3\wedge\neg y_1\wedge\neg y_2\wedge y_3) \vee (\neg x_1\wedge x_2\wedge\neg x_3\wedge\neg y_1\wedge y_2\wedge\neg y_3)$$

Symbolic safety analysis

- I : initial condition with parameters
  $x, x_2, \ldots\ldots, x_n$
- η : safe condition with parameters
  $y_1, y_2, \ldots\ldots, y_n$
- 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,
  - a risk state is reachable.
  - *the system is not safe.*

Symbolic safety analysis (backward)

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

- the state set as a proposition formula: $s(x_0,x_1,\ldots,x_n)$
- the risk state set as $r(x_0,x_1,\ldots,x_n)$
- the initial state set as $i(x_0,x_1,\ldots,x_n)$
- the transition set as $t(x_0,x_1,\ldots,x_n,x'_0,x'_1,\ldots,x'_n)$

$b_0 = r(x_0,x_1,\ldots,x_n) \wedge s(x_0,x_1,\ldots,x_n)$; $k = 1$;

repeat

$b_k = b_{k-1}\vee\exists x'_0\exists x'_1\ldots\exists x'_n(t(x_0,x_1,\ldots,x_n,x'_0,x'_1,\ldots,x'_n)\wedge(b_{k-1}\uparrow))$;

$k = k + 1$;

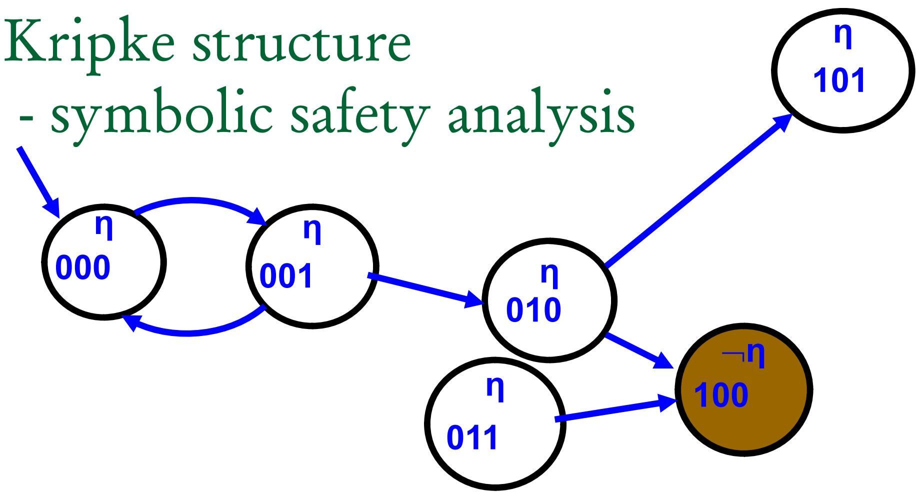
until $b_k \equiv b_{k-1}$;

if $(b_k \wedge i(x_0,x_1,\ldots,x_n)) \equiv$ false, return 'safe'; else return 'risky';

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

a least fixpoint procedure

Kripke structure - symbolic safety analysis

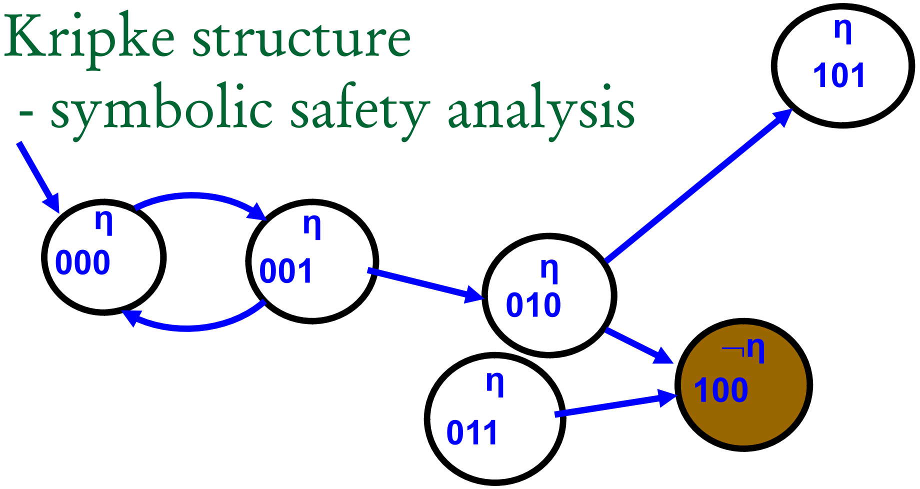


states: $s(x,y,z) \equiv (\neg x\wedge\neg y \wedge\neg z)\vee(\neg x\wedge\neg y \wedge z)\vee(\neg x\wedge y \wedge\neg z) \vee (\neg x\wedge y \wedge z)\vee(x\wedge\neg y \wedge\neg z)\vee(x\wedge\neg y \wedge z) \equiv (\neg x)\vee (x\wedge\neg y)$

initial state: $i(x,y,z)\equiv\neg x\wedge\neg y \wedge\neg z$

risk state: $r(x,y,z)\equiv x\wedge\neg y \wedge\neg z$

Kripke structure - symbolic safety analysis



transitions: $T(x,y,z,x',y',z') \equiv$
$(\neg x \wedge \neg y \wedge \neg z \wedge \neg x' \wedge \neg y' \wedge z') \vee (\neg x \wedge \neg y \wedge z \wedge \neg x' \wedge \neg y' \wedge \neg z')$
$\vee (\neg x \wedge \neg y \wedge z \wedge \neg x' \wedge y' \wedge \neg z') \vee (\neg x \wedge y \wedge \neg z \wedge x' \wedge \neg y' \wedge \neg z')$
$\vee (\neg x \wedge y \wedge \neg z \wedge x' \wedge \neg y' \wedge z') \vee (\neg x \wedge y \wedge z \wedge x' \wedge \neg y' \wedge \neg z')$

Symbolic safety analysis (backward)

$b_0 = r(x,y,z) \equiv x \wedge \neg y \wedge \neg z$

$b_1 = b_0 \vee \exists x' \exists y' \exists z'(t(x,y,z,x',y',z') \wedge b_0\uparrow)$

$= (x \wedge \neg y \wedge \neg z) \vee \exists x' \exists y' \exists z'(t(x,y,z,x',y',z') \wedge x' \wedge \neg y' \wedge \neg z')$

$= (x \wedge \neg y \wedge \neg z) \vee \exists x' \exists y' \exists z'(((\neg x \wedge y \wedge \neg z) \vee (\neg x \wedge y \wedge z)) \wedge x' \wedge \neg y' \wedge \neg z')$

$= (x \wedge \neg y \wedge \neg z) \vee (\neg x \wedge y \wedge \neg z) \vee (\neg x \wedge y \wedge z)$

$b_2 = b_1 \vee \exists x' \exists y' \exists z'(t(x,y,z,x',y',z') \wedge b_1\uparrow)$

$= (\neg x \wedge \neg y \wedge z) \vee (x \wedge \neg y \wedge \neg z) \vee (\neg x \wedge y \wedge \neg z) \vee (\neg x \wedge y \wedge z)$

$b_3 = b_2 \vee \exists x' \exists y' \exists z'(t(x,y,z,x',y',z') \wedge b_2\uparrow)$

$= (\neg x \wedge \neg y \wedge \neg z) \vee (\neg x \wedge \neg y \wedge z) \vee (x \wedge \neg y \wedge \neg z) \vee (\neg x \wedge y \wedge \neg z) \vee (\neg x \wedge y \wedge z)$

$b_4 = b_3 \vee \exists x' \exists y' \exists z'(t(x,y,z,x',y',z') \wedge b_3\uparrow)$

$= (\neg x \wedge \neg y \wedge \neg z) \vee (\neg x \wedge \neg y \wedge z) \vee (x \wedge \neg y \wedge \neg z) \vee (\neg x \wedge y \wedge \neg z) \vee (\neg x \wedge y \wedge z)$

$b_4 \wedge i(x,y,z) = (\neg x \wedge \neg y \wedge \neg z)$

fixpoint

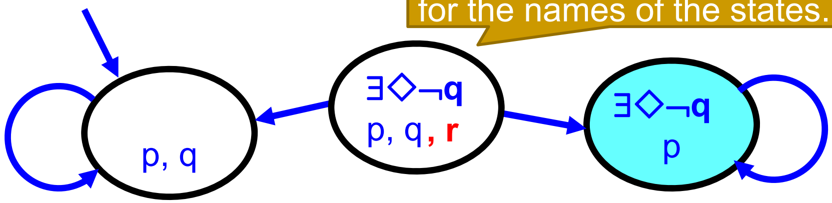
non-empty intersection with the initial condition → risk detected.

Symbolic safety analysis (backward)

One assumption for the correctness!

- Two states cannot be with the same proposition labeling.
- Otherwise, the collapsing of the states may cause problem.

may need a few propositions for the names of the states.



Symbolic safety analysis (forward)

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

- the state set as a proposition formula: $s(x_0,x_1,\ldots,x_n)$
- the risk state set as $r(x_0,x_1,\ldots,x_n)$
- the initial state set as $i(x_0,x_1,\ldots,x_n)$
- the transition set as $t(x_0,x_1,\ldots,x_n,x'_0,x'_1,\ldots,x'_n)$

change all primed variable to umprimed.

$f_0 = i(x_0,x_1,\ldots,x_n) \wedge s(x_0,x_1,\ldots,x_n)$; $k = 1$;

repeat

$f_k = f_{k-1} \vee (\exists x_0 \exists x_1 \ldots \exists x_n(t(x_0,x_1,\ldots,x_n,x'_0,x'_1,\ldots,x'_n) \wedge f_{k-1}))\downarrow$;

$k = k+1$;

until $f_k \equiv f_{k-1}$;

if $(f_k \wedge r(x_0,x_1,\ldots,x_n)) \equiv$ false, return 'safe'; else return 'risky';

Symbolic safety analysis (forward)

$f_0 = i(x,y,z) \equiv \neg x\wedge\neg y \wedge\neg z$

$f_1 = f_0 \vee (\exists x\exists y\exists z(t(x,y,z,x',y',z') \wedge f_0))\downarrow$

$= (\neg x\wedge\neg y \wedge\neg z)\vee (\exists x\exists y\exists z(t(x,y,z,x',y',z') \wedge \neg x\wedge\neg y\wedge\neg z))\downarrow$

$= (\neg x\wedge\neg y \wedge\neg z)\vee(\exists x\exists y\exists z(\neg x'\wedge\neg y'\wedge z'\wedge \neg x\wedge\neg y\wedge\neg z))\downarrow$

$= (\neg x\wedge\neg y \wedge\neg z)\vee(\neg x'\wedge\neg y'\wedge z')\downarrow$

$= (\neg x\wedge\neg y \wedge\neg z)\vee(\neg x\wedge\neg y\wedge z) = \neg x\wedge\neg y$

$f_2 = f_1 \vee (\exists x\exists y\exists z(t(x,y,z,x',y',z') \wedge f_1)\downarrow = (\neg x\wedge\neg y)\vee(\neg x\wedge y\wedge\neg z)$

$f_3 = f_2 \vee (\exists x\exists y\exists z(t(x,y,z,x',y',z') \wedge f_2)\downarrow = (\neg y)\vee(\neg x\wedge y\wedge\neg z)$

$f_4 = f_3 \vee (\exists x\exists y\exists z(t(x,y,z,x',y',z') \wedge f_3)\downarrow = (\neg y)\vee(\neg x\wedge y\wedge\neg z)$

$f_4 \wedge r(x,y,z) = ((\neg y)\vee(\neg x\wedge y\wedge\neg z)) \wedge(x\wedge\neg y\wedge\neg z) = (x\wedge\neg y\wedge\neg z)$

fixpoint

non-empty intersection with the risk condition → risk detected.

Bounded model-checking

Encode the states with variables $x_{0,k},x_{1,k},\ldots,x_{n,k}$.

The value of x_n at state k.

- the state set as a proposition formula: $s(x_{0,k},x_{1,k},\ldots,x_{n,k})$
- the risk state set as $r(x_{0,k},x_{1,k},\ldots,x_{n,k})$
- the initial state set as $i(x_{0,0},x_{1,0},\ldots,x_{n,0})$
- the transition set as $t(x_{0,k-1},x_{1,k-1},\ldots,x_{n,k-1},x_{0,k},x_{1,k},\ldots,x_{n,k})$

$f_0 = i(x_{0,0},x_{1,0},\ldots,x_{n,0}) \wedge s(x_{0,0},x_{1,0},\ldots,x_{n,0})$; $k = 1$;

repeat

$f_k = t(x_{0,k-1},x_{1,k-1},\ldots,x_{n,k-1},x_{0,k},x_{1,k},\ldots,x_{n,k})\wedge f_{k-1}$;

$k = k + 1$;

until $f_k\wedge r(x_{0,k},x_{1,k},\ldots,x_{n,k}) \neq$ false

When to stop ?
1. diameter of the state graph
2. explosion up to tens of steps.

Bounded model-checking

$f_0 = i(x,y,z) \equiv \neg x_0\wedge\neg y_0 \wedge\neg z_0$

$f_1 = t(x_0,y_0,z_0,x_1,y_1,z_1) \wedge f_0 = \neg x_0\wedge\neg y_0\wedge\neg z_0\wedge\neg x_1\wedge\neg y_1\wedge z_1$

$f_2 = t(x_1,y_1,z_1,x_2,y_2,z_2) \wedge f_1$

$= \neg x_0\wedge\neg y_0\wedge\neg z_0\wedge\neg x_1\wedge\neg y_1\wedge z_1\wedge((\neg x_2\wedge\neg y_2\wedge\neg z_2)\vee(\neg x_2\wedge y_2\wedge\neg z_2))$

$f_3 = t(x_2,y_2,z_2,x_3,y_3,z_3) \wedge f_2$

$= \neg x_0\wedge\neg y_0\wedge\neg z_0\wedge\neg x_1\wedge\neg y_1\wedge z_1$

$\wedge((\neg x_2\wedge\neg y_2\wedge\neg z_2\wedge\neg x_3\wedge\neg y_3\wedge z_3)$

$\vee(\neg x_2\wedge y_2\wedge\neg z_2\wedge((x_3\wedge\neg y_3\wedge\neg z_3)\vee(x_3\wedge\neg y_3\wedge z_3)))$

$)$

$= \neg x_0\wedge\neg y_0\wedge\neg z_0\wedge\neg x_1\wedge\neg y_1\wedge z_1$

$\wedge((\neg x_2\wedge\neg y_2\wedge\neg z_2\wedge\neg x_3\wedge\neg y_3\wedge z_3)\vee(\neg x_2\wedge y_2\wedge\neg z_2\wedge x_3\wedge\neg y_3))$

$f_3 \wedge r(x_3,y_3,z_3) = (x_3\wedge\neg y_3\wedge\neg z_3)$

Symbolic liveness analysis

Encode the states with variables x0,x1,…,xn.

- the state set as a proposition formula: $s(x_0,x_1,\ldots,x_n)$
- the non-liveness state set as $b(x_0,x_1,\ldots,x_n)$
- the initial state set as $i(x_0,x_1,\ldots,x_n)$
- the transition set as $t(x_0,x_1,\ldots,x_n,x'_0,x'_1,\ldots,x'_n)$

$b_0 = b(x_0,x_1,\ldots,x_n) \wedge s(x_0,x_1,\ldots,x_n)$; $k = 1$;

repeat

$b_k = b_{k-1}\wedge\exists x'_0\exists x'_1\ldots\exists x'_n(t(x_0,x_1,\ldots,x_n,x'_0,x'_1,\ldots,x'_n)\wedge b_{k-1}\uparrow)$;

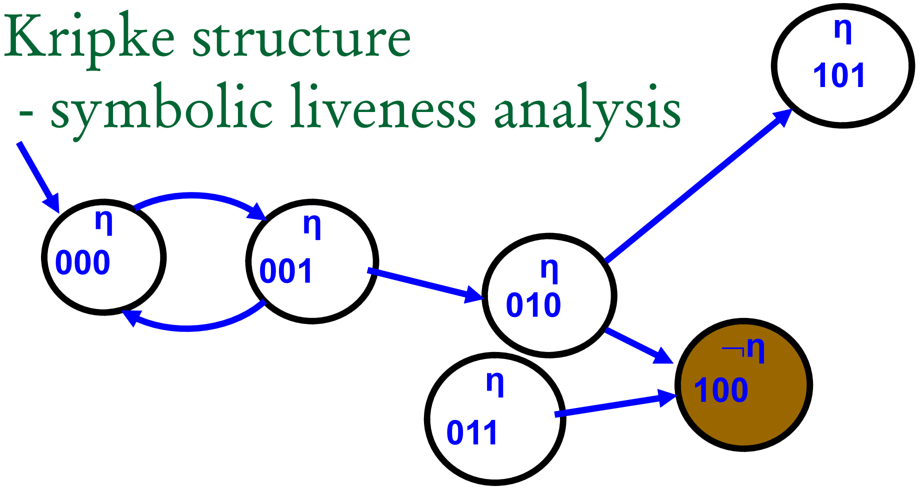
$k = k + 1$;

until $b_k \equiv b_{k-1}$;

if $(b_k \wedge i(x_0,x_1,\ldots,x_n)) \equiv$ false, return 'live'; else return 'not live';

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

Kripke structure
- symbolic liveness analysis

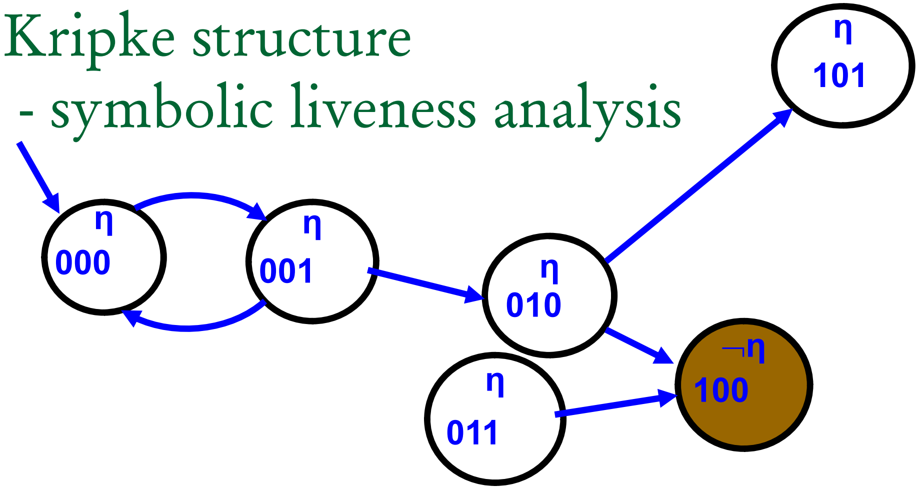


states: $s(x,y,z) \equiv (\neg x \wedge \neg y \wedge \neg z) \vee (\neg x \wedge \neg y \wedge z) \vee (\neg x \wedge y \wedge \neg z)$
$\vee (\neg x \wedge y \wedge z) \vee (x \wedge \neg y \wedge \neg z) \vee (x \wedge \neg y \wedge z)$
$\equiv (\neg x) \vee (x \wedge \neg y)$

initial state: $i(x,y,z) \equiv \neg x \wedge \neg y \wedge \neg z$

non-liveness state: $b(x,y,z) \equiv (\neg x) \vee (x \wedge \neg y \wedge z)$

Kripke structure
- symbolic liveness analysis



transitions: $T(x,y,z,x',y',z') \equiv$
$(\neg x \wedge \neg y \wedge \neg z \wedge \neg x' \wedge \neg y' \wedge z') \vee (\neg x \wedge \neg y \wedge z \wedge \neg x' \wedge \neg y' \wedge \neg z')$
$\vee (\neg x \wedge \neg y \wedge z \wedge \neg x' \wedge y' \wedge \neg z') \vee (\neg x \wedge y \wedge \neg z \wedge x' \wedge \neg y' \wedge \neg z')$
$\vee (\neg x \wedge y \wedge \neg z \wedge x' \wedge \neg y' \wedge z') \vee (\neg x \wedge y \wedge z \wedge x' \wedge \neg y' \wedge \neg z')$

Symbolic liveness analysis

$b0 = b(x,y,z) \equiv (\neg x) \vee (x \wedge \neg y \wedge z)$

$b1 = b0 \wedge \exists x' \exists y' \exists z' (T(x,y,z,x',y',z') \wedge b0')$

$= ((\neg x) \vee (x \wedge \neg y \wedge z))$
$\wedge \exists x' \exists y' \exists z' (T(x,y,z,x',y',z') \wedge ((\neg x') \vee (x' \wedge \neg y' \wedge z')))$

$= ((\neg x) \vee (x \wedge \neg y \wedge z)) \wedge$
$\exists x' \exists y' \exists z' (((\neg x \wedge \neg y \wedge \neg z) \vee (\neg x \wedge y \wedge \neg z) \vee (\neg x \wedge \neg y \wedge z))$
$\wedge ((\neg x') \vee (x' \wedge \neg y' \wedge z')))$

$= (\neg x \wedge \neg y \wedge \neg z) \vee (\neg x \wedge y \wedge \neg z) \vee (\neg x \wedge \neg y \wedge z)$

$b2 = b1 \wedge \exists x' \exists y' \exists z' (T(x,y,z,x',y',z') \wedge b1')$

$= (\neg x \wedge \neg y \wedge \neg z) \vee (\neg x \wedge y \wedge \neg z)$

$b3 = b2 \wedge \exists x' \exists y' \exists z' (T(x,y,z,x',y',z') \wedge b2')$

$= (\neg x \wedge \neg y \wedge \neg z) \vee (\neg x \wedge y \wedge \neg z)$

fixpoint

non-empty intersection with the initial condition → non-liveness detected.

CTL
- symbolic model-checking algorithm

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

label(φ) {

case p, return p;

case ¬φ, return ¬label(φ);

case φ∨ψ, return label(φ) ∨ label(ψ),

case ∃○φ, return $\vee_{i=n}$ *pred*(r_i, label(φ));

case ∃ ψ_1 **U** ψ_2 , return lfp(l abel(ψ_1), label(ψ_2));

case ∃□φ, return gfp(label(φ));

}

Symbolic model-checking
- with real-world programs

Consider guarded commands with modes (GCM)

Guard → Actions

- Guard is a propositional formula of state variables.
- Actions is a command of the following syntax.

C ::= ACT | *{C}* | C C | *if (*B*)* C *else* C | *while (*B*)* C

ACT ::= *;* | *goto name;* | *x* = E *;*

Guarded commands with modes (GCM)

program

```
1:  w = 0;
2:  x = 0;
3:  y = z*z;
4:  while (x < y) {
5:    w = w + x*z;
6:    x = x + 1;
7:  }
8:  if (w > z*z*z) w = z*z*z;
```

guarded commands

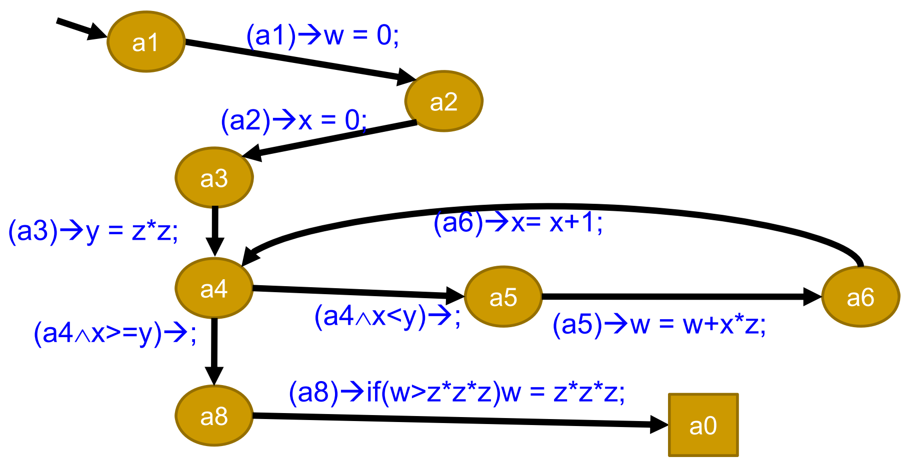
```
(pc==1) → w = 0; pc=2;
(pc==2) → x = 0; pc=3;
(pc==3) → y = z*z; pc=4;
(pc==4&&x>=y) → pc=8;
(pc==4&&x < y) → pc=5;
(pc==5) → w=w+x*z; pc=6;
(pc==6) → x=x+1; pc=4;
(pc==8) → if (w>z*z*z) w= z*z*z;
```


A state-transition
- represented as a GCM

8 rules in total:

(a1) → w = 0; goto a2;
(a2) → x = 0; goto a3;
(a3) → y = z*z; goto a4;
(a4&&x>=y) → goto a8;
(a4&&x < y) → goto a5;
(a5) → w=w+x*z; goto a6;
(a6) → x=x+1; goto a4;
(a8) → if (w>z*z*z) w= z*z*z; }

A state-transition
- represented as a GCM



Transition relation - from state-transition graphs

Given a set of rules $r_1, r_2, \ldots, r_m$ of the form

r_k: $(\tau_k) \rightarrow y_{k,0}=d_0; y_{k,1}=d_1; \ldots; y_{k,nk}=d_{nk};$

$$t(x_0,x_1,\ldots,x_n,x'_0,x'_1,\ldots,x'_n) \equiv \bigvee_{k\in[1,m]} \Big(\tau_k \wedge y'_{k,0}==d_0 \wedge y'_{k,1}==d_1 \wedge \ldots \wedge y'_{k,nk}==d_{nk} \wedge \bigwedge_{h\in[1,n]} (x_h \notin \{y_{k,0}, y_{k,1}, \ldots, y_{k,nk}\} \Rightarrow x_h == x'_h) \Big)$$

Transition relation from GCM rules.

Given a set of rules for X={x,y,z}

r_1: (x<y&& y>2) → y=x+y; x=3;

r_2: (z>=2) → y=x+1; z=0;

r_3: (x<2) → x=0;

$$t(x_0,x_1,\ldots,x_n,x'_0,x'_1,\ldots,x'_n) \equiv (x<y \wedge y>2 \wedge y'==x+y \wedge x'==3 \wedge z'==z) \vee (z>=2 \wedge y'==x+1 \wedge z'==0 \wedge x'==x) \vee (x<2 \wedge x'==0 \wedge y'==y \wedge z'==z)$$

Transition relation - from state-transition graphs

In gneral, transition relation is expensive to construct.

Can we do the following state-space construction

$\exists x'_0 \exists x'_1 \ldots \exists x'_n (t(x_0,x_1,\ldots,x_n,x'_0,x'_1,\ldots,x'_n) \wedge (b_{k-1}\uparrow))$

directly with the GCM rules ?

Yes, ***on-the-fly state space construction***.

On-the-fly precondition calculation with GCM rules.

Given a set of rules $r_1, r_2, \ldots, r_m$ of the form

r_k: $(\tau_k) \rightarrow y_{k,0}=d_0; y_{k,1}=d_1; \ldots; y_{k,nk}=d_{nk};$

$$\exists x'_0 \exists x'_1 \ldots \exists x'_n (t(x_0,x_1,\ldots,x_n,x'_0,x'_1,\ldots,x'_n) \wedge (b\uparrow)) \equiv \bigvee_{k\in[1,m]} \Big(\tau_k \wedge \exists y_{k,0} \exists y_{k,1} \ldots \exists y_{k,nk} \big(b \wedge \bigwedge_{h\in[0,nk]} y_{k,h}==d_h\big) \Big)$$

However, GCM rules are more complex than that.

On-the-fly precondition calculation with GCM rules.

Given a set of rules for X={x,y,z}

r_1: (x<y&& y>2) → y=z; x=3;

r_2: (z>=2) → y=x+1; z=7;

r_3: (x<2) → z=0;

B

$$\exists x'_0 \exists x'_1 \ldots \exists x'_n (t(x_0,x_1,\ldots,x_n,x'_0,x'_1,\ldots,x'_n) \wedge (x<4 \wedge z>5)\uparrow)$$

$$\equiv (x<y \wedge y>2 \wedge \exists y \exists x(x<4 \wedge z>5 \wedge y==z \wedge x==3))$$
$$\vee (z>=2 \wedge \exists y \exists z(x<4 \wedge z>5 \wedge y==x+1 \wedge z==7))$$
$$\vee (x<2 \wedge \exists z(x<4 \wedge z>5 \wedge z==0))$$
$$\equiv (x<y \wedge y>2 \wedge z>5) \vee (z>=2 \wedge x<4) \vee (x<2 \wedge \exists z(false))$$
$$\equiv (x<y \wedge y>2 \wedge z>5) \vee (z>=2 \wedge x<4)$$

On-the-fly precondition calculation with GCM rules.

Given a set of rules $r_1, r_2, \ldots, r_m$ of the form

r_k: (τ_k) → s_k;

$$\exists x'_0 \exists x'_1 \ldots \exists x'_n (t(x_0,x_1,\ldots,x_n,x'_0,x'_1,\ldots,x'_n) \wedge (b\uparrow))$$
$$\equiv \bigvee_{k\in[1,m]} \left(\tau_k \wedge pre(s_k, b)\right)$$

precondition procedure

A general propositional formula

What is pre(s, b) ?

A GCM statement

On-the-fly precondition calculation with GCM rules.

Given a set of rules $r_1, r_2, \ldots, r_m$ of the form

r_k: (τ_k) → s_k;

What is pre(s, b) ?

new expression obtained from b by replacing every occurrence of x with E.

- pre($x = E$;, b) ≡ b[x/E]

Ex 1. the precondition to x=x+z;
(x==y+2 ∧ x<4∧z>5) [x/x+z] ≡ x+z==y+2 ∧ x+z<4∧z>5

Ex 2. the precondition to x=5;
(x==y+2 ∧ x<4∧z>5) [x/x+z] ≡ 5==y+2 ∧ 5<4∧z>5

Ex 3. the precondition to x=2*x+1;
(x==y+2 ∧ x<4∧z>5) [x/x+z] ≡ 2*x+1==y+2 ∧ 2*x+1<4∧z>5

On-the-fly precondition calculation with GCM rules.

Given a set of rules $r_1, r_2, \ldots, r_m$ of the form

r_k: (τ_k) → s_k;

What is pre(s, b) ?

new expression obtained from b by replacing every occurrence of x with E.

- pre($x = E$;, b) ≡ b[x/E]
- pre($s_1 s_2$, b) ≡ pre(s_1, pre(s_2, b))
- pre(if (B) $s_1$else s_2) ≡ (B∧pre(s_1, b))∨(¬B∧pre(s_2,b))
- pre(while (B) s, b) ≡ ….

Ex. the precondition to x=x+z;
(x==y+2 ∧ x<4∧z>5) [x/x+z]
≡ x+z==y+2 ∧ x+z<4∧z>5

On-the-fly precondition calculation with GCM rules.

Given a set of rules $r_1, r_2, \ldots, r_m$ of the form

r_k: $(\tau_k) \rightarrow s_k$;

What is pre(s,b) ?

pre(while (B) s, b) $\equiv$ formula $L_1 \vee L_2$ for

L_1: those states that reach $\neg B \wedge b$ with finite steps of s through states in B; and

L_2: those states that never leave B with steps of s.

On-the-fly precondition calculation with GCM rules.

L_1: those states that reach $\neg B \wedge b$ with finite steps of s through states in B

also a least fixpoint procedure

```
w0 = ¬B∧b; k = 1;
repeat
   wk = wk-1∨(B∧pre(s, wk-1));
   k = k +1;
 until wk ≡ wk-1;
return wk;
```


Precondition to b through while (B) s;

```
w0 = ¬B∧b; k = 1;
repeat
   wk = wk-1∨(B∧pre(s, wk-1));
   k = k +1;
 until wk ≡ wk-1;
return wk;
```

Example: $b \equiv x{=}{=}2 \wedge y == 3$

while (x < y) x = x+1;

L1 computation.

$w_0 \equiv x>=y \wedge x==2 \wedge y==3 \equiv$ false ; k = 1;

$w_1 \equiv$ false$\vee$(x<y$\wedge$pre(x=x+1, false));

$\equiv$ false$\vee$(x<y$\wedge$false);

$\equiv$ false;

On-the-fly precondition calculation with GCM rules.

Given a set of rules $r_1, r_2, \ldots, r_m$ of the form

pre(while (B) s, b)

L_2: those states that never leave B with steps of s.

a *greatest* fixpoint procedure

```
w0 = B; k = 1;
repeat
   wk = wk-1∧pre(s, wk-1);
   k = k +1;
 until wk ≡ wk-1;
return wk;
```

Precondition to b through while (B) s;

$w_0 = B$; $k = 1$;
repeat
  $w_k = w_{k-1} \wedge pre(s, w_{k-1})$;
  $k = k + 1$;
until $w_k \equiv w_{k-1}$;
return w_k;

Example:

while (x<y && x>0) x = x+1;

L2 computation.

$w_0 \equiv x<y \wedge x>0$; $k = 1$;

$w_1 \equiv x<y \wedge x>0 \wedge pre(x=x+1, x<y \wedge x>0)$

$\equiv x<y \wedge x>0 \wedge x+1<y \wedge x+1>0 \equiv x>0 \wedge x+1<y$

$w_2 \equiv x+1<y \wedge x>0 \wedge pre(x=x+1, x+1<y \wedge x>0)$

$\equiv x+1<y \wedge x>0 \wedge x+2<y \wedge x+1>0 \equiv x>0 \wedge x+2<y$

non-terminating for algorithms and protocols!

Precondition to b through while (B) s;

$w_0 = B$; $k = 1$;
repeat
  $w_k = w_{k-1} \wedge pre(s, w_{k-1})$;
  $k = k + 1$;
until $w_k \equiv w_{k-1}$;
return w_k;

Example:

while (x>y && x>0) x = x+1;

L2 computation.

$w_0 \equiv x>y \wedge x>0$; $k = 1$;

$w_1 \equiv x>y \wedge x>0 \wedge pre(x=x+1, x>y \wedge x>0)$

$\equiv x>y \wedge x>0 \wedge x+1>y \wedge x+1>0 \equiv x>y \wedge x>0$

terminating for algorithms and protocols!

Precondition to b through while (B) s;

$w_0 = \neg B \wedge b$; $k = 1$;
repeat
  $w_k = w_{k-1} \vee (B \wedge pre(s, w_{k-1}))$;
  $k = k + 1$;
until $w_k \equiv w_{k-1}$;
return w_k;

Example: $b \equiv x==2 \wedge y==3$

while (x>y && x>0) x = x+1;

L1 computation.

$w_0 \equiv (x<=y \vee x<=0) \wedge x==2 \wedge y==3 \equiv x==2 \wedge y==3$;

$w_1 \equiv (x==2 \wedge y==3) \vee (x>y \wedge x>0 \wedge pre(x=x+1, x==2 \wedge y==3))$;

$\equiv (x==2 \wedge y==3) \vee (x>y \wedge x>0 \wedge x==1 \wedge y==3)$;

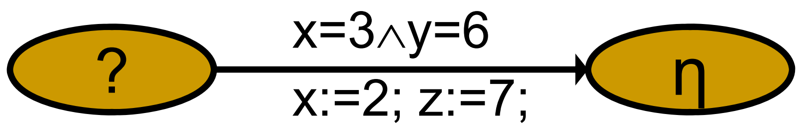
$\equiv (x==2 \wedge y==3) \vee false$

$\equiv x==2 \wedge y==3$

Symbolic weakest precondition

Assume program with rules

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



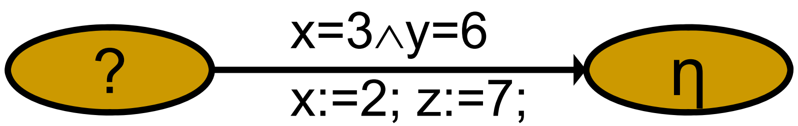
- x, y, z are discrete variables with range declarations

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 ?*

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

Symbolic safety analysis - with Kripke structures as programs

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

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

```
lfp (φ, ψ) /* for ∃φUψ*/ {
   Z' := false; Z:= ψ;
   while (Z ≠ Z') {
     Z' := Z;
     Z := Z ∨ (φ∧∨i=n pred(ri, Z));
   }
   return (Z);
}
```

$I \wedge lfp(true, \neg\eta) \neq false$

Initial condition

risk predicate

Symbolic liveness analysis - with Kripke structures as programs

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

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

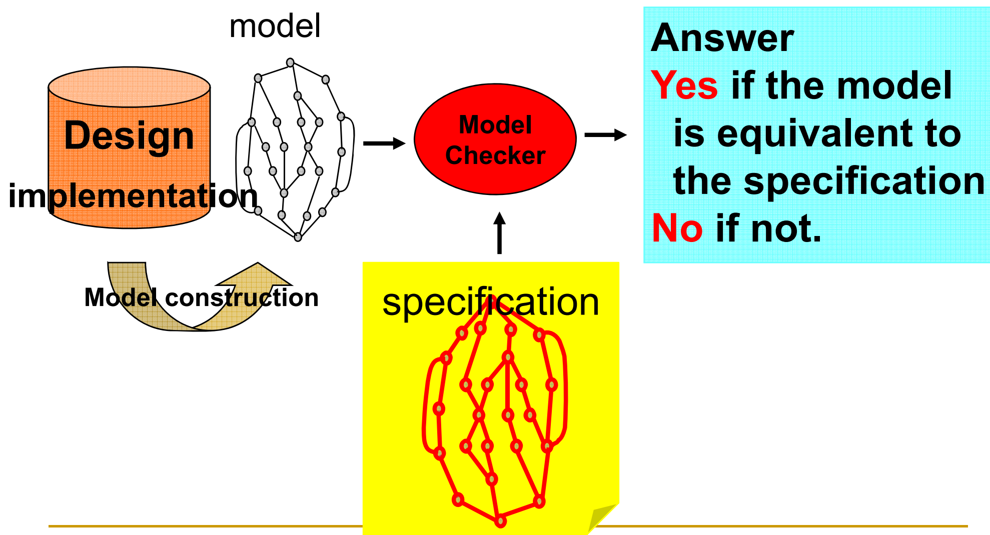
```
gfp (φ) /* for ∃□φ */{
   Z' := false; Z:= φ;
   while (Z ≠ Z') {
     Z' := Z;
     Z:= φ ∧∨i=n pred(ri, Z);
   }
   return (Z);
}
```

$I \wedge gfp(\neg\eta) \neq \varnothing$

Initial condition

negative liveness predicate

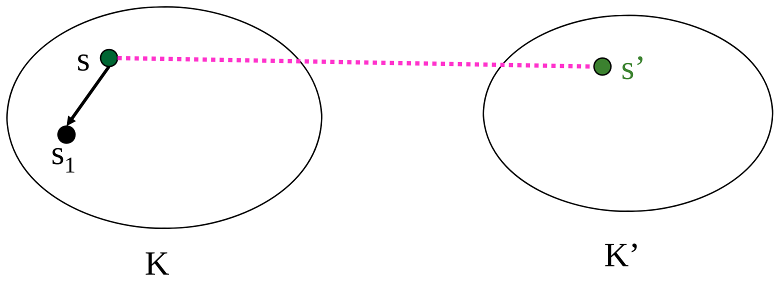
Bisimulation Framework



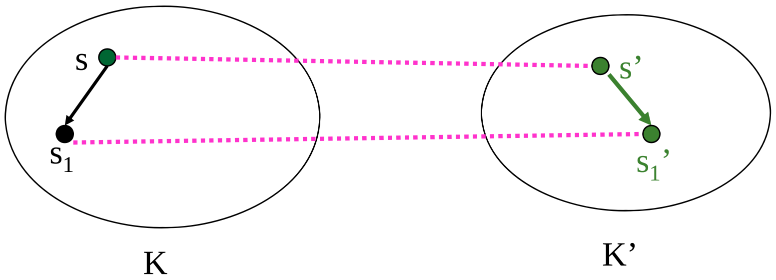
Bisimulation-checking

- 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∈S×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)

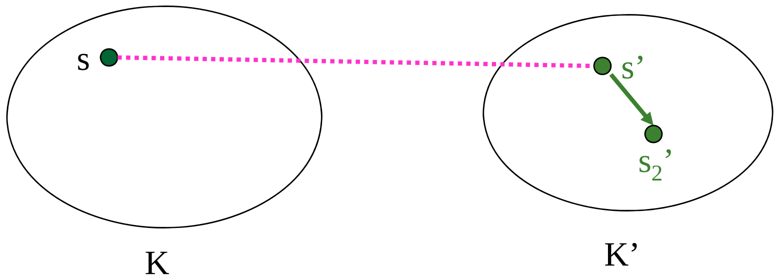
Bisimulations



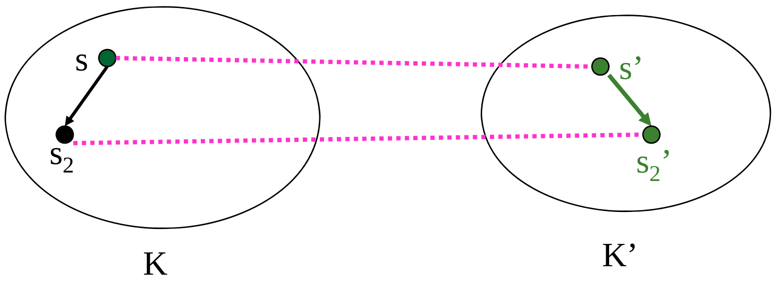
Bisimulations



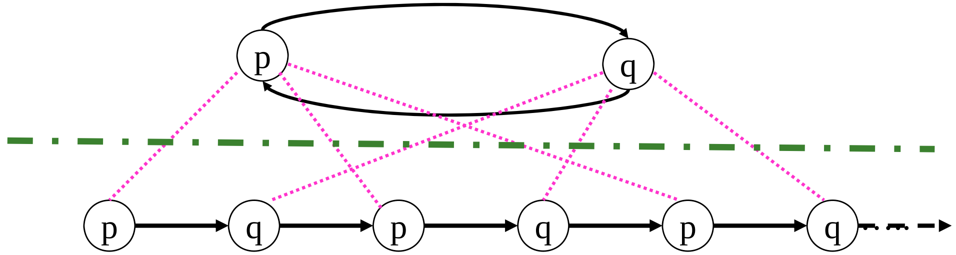
Bisimulations



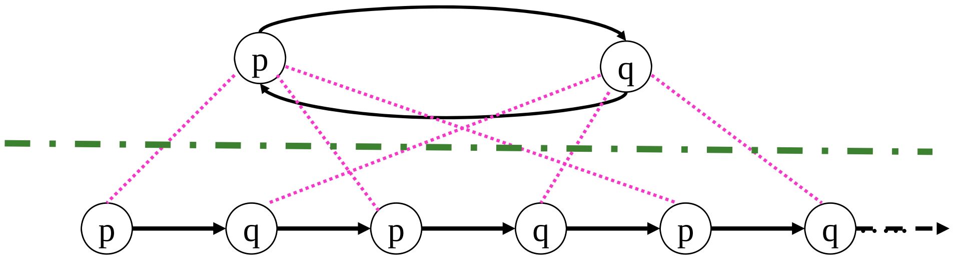
Bisimulations



Examples

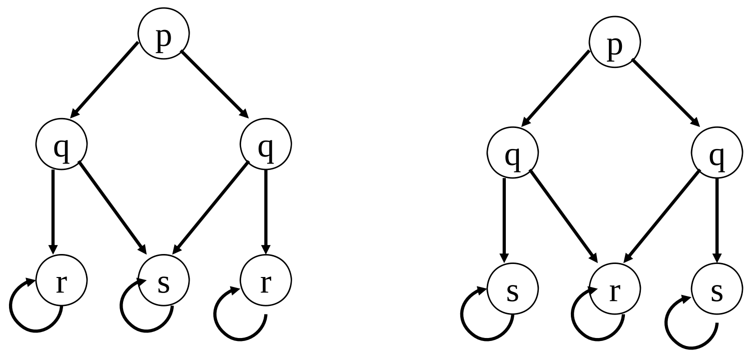


Examples

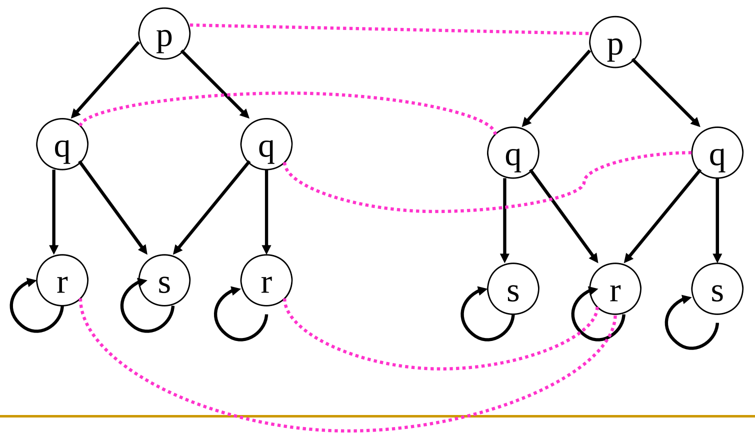


Unwinding preserves bisimulation

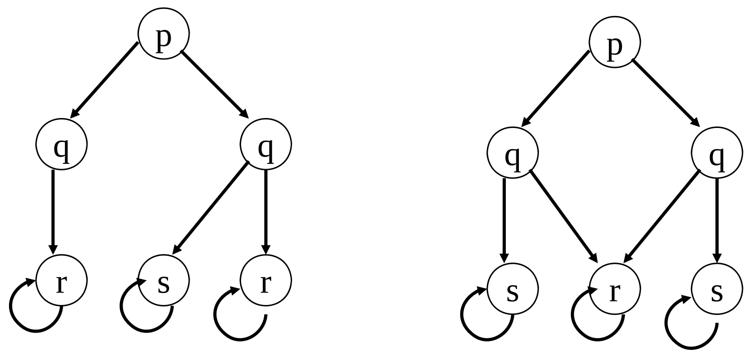
Examples



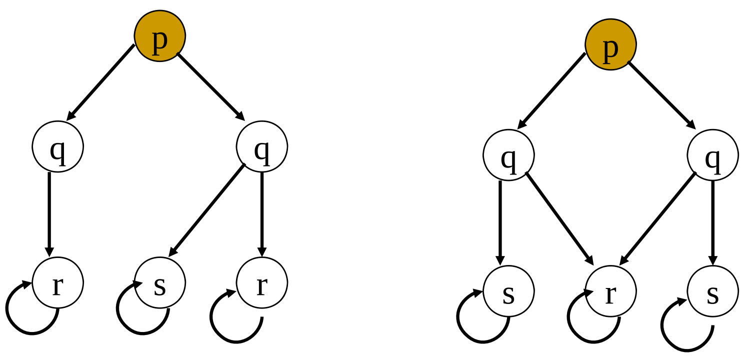
Examples



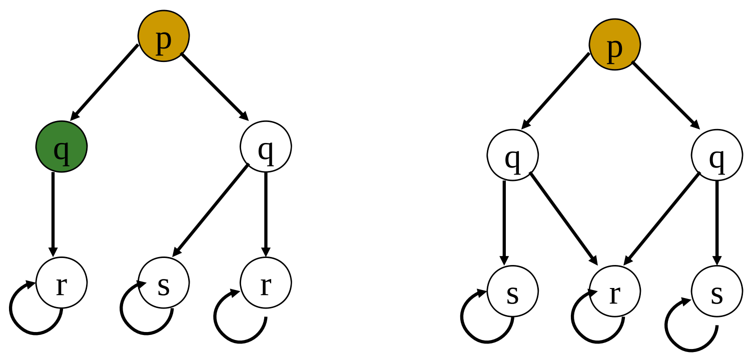
Examples



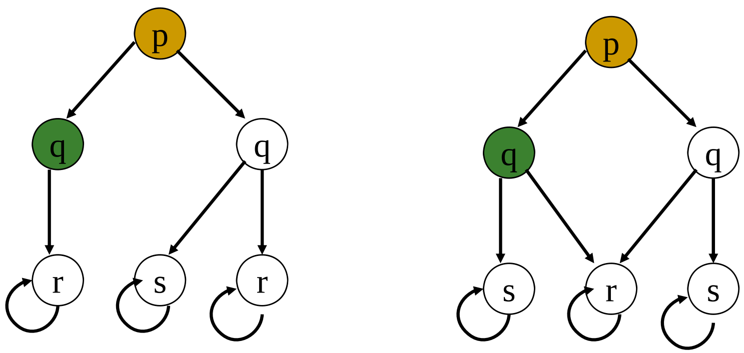
Examples



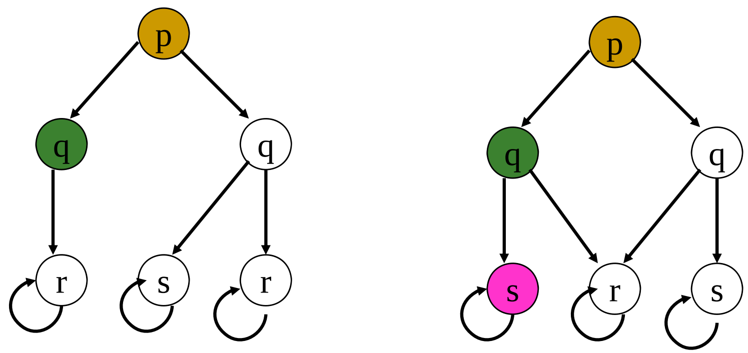
Examples



Examples



Examples



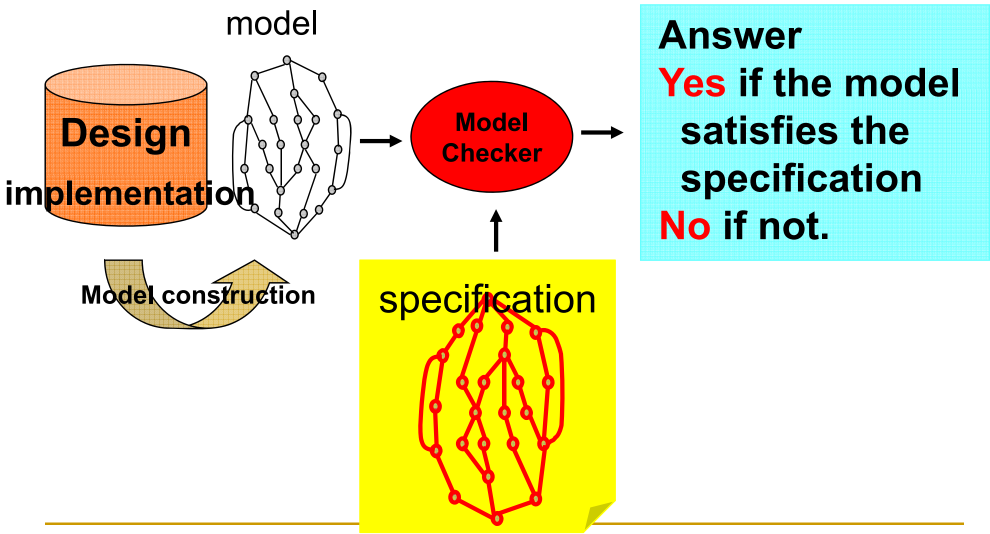
Bisimulations

- 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 $B \subseteq S \times S'$ 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')
- $B \subseteq S \times S'$, a bisimulation.
- Suppose $B(s, s')$.
- FACT: For any CTL formula ψ (over AP), $K,s \models \psi$ iff $K',s' \models \psi$.
- 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.
- $B \in S \times S'$ 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 $B \subseteq S \times S'$ between K and K' such that for each s_0 in S_0 there exists s_0' in S_0' such that $B(s_0, s_0')$.

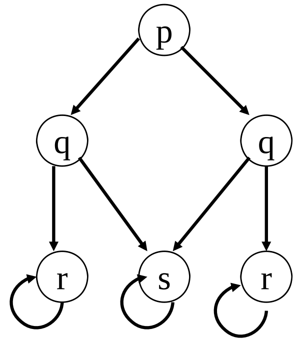
Bisimulation Quotients

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

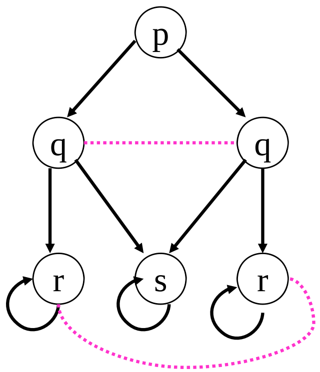
Bisimulation Quotient

- K = (S, S_0, R, AP, L)
- [s] = {s' | s R s'}.
- K' = K / R = (S', S'_0, R', AP,L').
  - S' = {[s] | s 2 S}
  - S'_0 = {[s_0] | s_0 2 S_0}
  - R' = {([s], [s']) | R(s_1, s_1') , s_1∈[s], s_1'∈[s']}
  - L'([s]) = L(s).

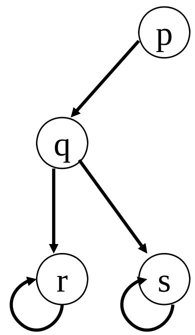
Examples



Examples



Examples



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

Kripke structure
- simulation-checking

/* Given model A = (S, S_0, R, L), spec. A'=(S', S'_0, R', L') */
Simulation-checking(A,A') /* using greatest fixpoint algorithm */ {
    Let B={(s,s') | s∈S, s'∈S', L(s)=L'(s')} ;
    repeat {
        B = B - {(s,s') | (s,s')∈B, ∃(s,t)∈R∀(s',t')∈R'((t,t')∉B)};
    } until no more changes to B.
    if there is an $s_0 \in S_0$ with $\forall s'_0 \in S'_0((s_0,s'_0) \notin B)$,
        return 'no simulation,'
        else return 'simulation exists.'
}
The procedure terminates since B is finite in the Kripke structure.

Kripke structure
- bisimulation-checking

/* Given model A = (S, S_0, R, L), spec. A'=(S', S'_0, R', L') */
Bisimulation-checking(A,A') /* using greatest fixpoint algorithm */ {
    Let B={(s,s') | s∈S, s'∈S', L(s)=L'(s')} ;
    repeat {
        B = B - {(s,s') | (s,s')∈B, ∃(s,t)∈R∀(s',t')∈R'((t,t')∉B)};
        B = B - {(s,s') | (s,s')∈B, ∃(s',t')∈R'∀(s,t)∈R((t,t')∉B)};
    } until no more changes to B.
    if there is an $s_0 \in S_0$ with $\forall s'_0 \in S'_0((s_0,s'_0) \notin B)$,
        return 'no simulation,'
    if there is an $s'_0 \in S'_0$ with $\forall s_0 \in S_0((s_0,s'_0) \notin B)$,
        return 'no simulation,'
    else return 'simulation exists.'
}

(Bi)Simulation
- complexities

- Bisimulation: O((m+n)log(m+n))
- Simulation: O(m n)
- In contrast, finding a graph homeomorphism is NP-complete.

Symbolic simulation-checking

- Encode the states with variables
  - $x_0, x_1, \ldots, x_n$ (for the model) and
  - $y_0, y_1, \ldots, y_m$. (for the spec.)

  Usually there are shared variables
  between $\{x_0, x_1, \ldots, x_n\}$ and $\{y_0, y_1, \ldots, y_m\}$.
  L(s)=L'(s') means that the shared variables are of the same values.
- the state sets as proposition formulas:
  - $s(x_0, x_1, \ldots, x_n)$ & $s(y_0, y_1, \ldots, y_m)$
- the initial state set as
  - $i(x_0, x_1, \ldots, x_n)$ & $i'(y_0, y_1, \ldots, y_m)$
- the transition set as
  - $R(x_0, x_1, \ldots, x_n, x'_0, x'_1, \ldots, x_n)$ & $R'(y_0, y_1, \ldots, y_n, y'_0, y'_1, \ldots, y'_n)$

Symbolic simulation-checking

$B_0 = \bigwedge_{L(x0,x1,\ldots,xn) = L(y0,y1,\ldots,ym)} s(x_0, x_1, \ldots, x_n) \wedge s(y_0, y_1, \ldots, y_m)$;

for ($k = 1$, B_1= false; $B_k \neq B_{k-1}$; $k=k+1$)

$B_k = B_{k-1} \wedge \neg \exists x'_0 \exists x'_1 \ldots \exists x'_n$ (

$R(x_0, x_1, \ldots, x_n, x'_0, x'_1, \ldots, x'_n)$

$\wedge \neg \exists y'_0 \exists y'_1 \ldots \exists y'_m$ (

$R'(y_0, y_1, \ldots, y_m, y'_0, y'_1, \ldots, y'_m) \wedge (B_{k-1} \uparrow)$

)) ;

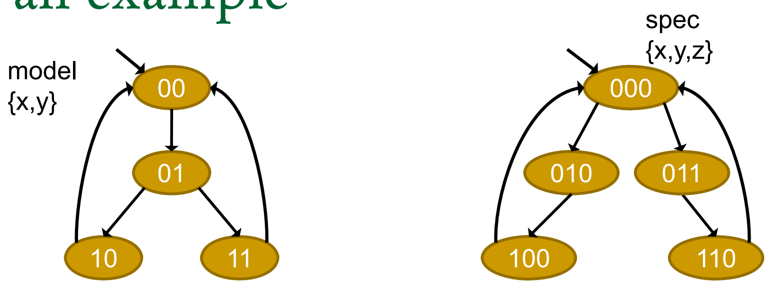
if $(i(x_0, x_1, \ldots, x_n) \neq \exists y_0 \exists y_1 \ldots \exists y_m (B_k))$,

return 'no simulation';

else return 'a simulation exists';

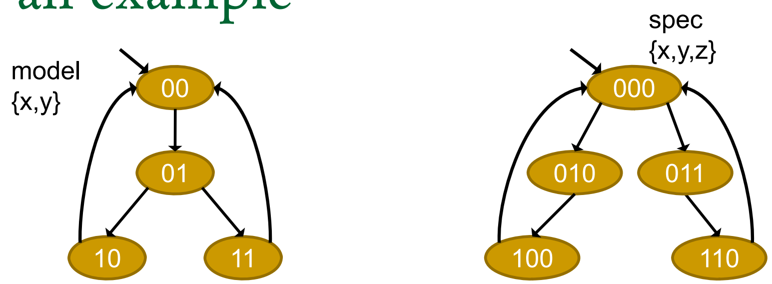
change all umprimed variable in B_{k-1} to primed.

Symbolic simulation-checking - an example



- $s(x,y) \equiv \text{true}$, $s'(x,y,z) \equiv \neg z \vee (\neg x \wedge y \wedge z)$
- $i(x,y) \equiv \neg x \wedge \neg y$, $i'(x,y,z) \equiv \neg x \wedge \neg y \wedge \neg z$
- $R(x,y,x',y') \equiv \ldots\ldots$, $R'(x,y,z,x',y',z') \equiv \ldots\ldots$

Symbolic simulation-checking - an example



- $R(x,y,x',y') \equiv (\neg x \wedge \neg y \wedge \neg x' \wedge y') \vee (\neg x \wedge y \wedge x' \wedge \neg y')$
  $\vee (\neg x \wedge y \wedge x' \wedge y') \vee (x \wedge \neg y \wedge \neg x' \wedge \neg y') \vee (x \wedge y \wedge \neg x' \wedge \neg y')$
- $R'(x,y,z,x',y',z') \equiv (\neg x \wedge \neg y \wedge \neg z \wedge \neg x' \wedge y')$
  $\vee (\neg x \wedge y \wedge \neg z \wedge x' \wedge \neg y' \wedge \neg z') \vee (\neg x \wedge y \wedge z \wedge x' \wedge y' \wedge \neg z')$
  $\vee (x \wedge \neg y \wedge \neg z \wedge \neg x' \wedge \neg y' \wedge \neg z') \vee (x \wedge y \wedge \neg z \wedge \neg x' \wedge \neg y' \wedge \neg z')$

Symbolic simulation-checking - an example

$B_0 = s(x,y) \land s'(x,y,z) = \neg z \lor (\neg x \land y \land z)$

$B_1 = (\neg z \lor (\neg x \land y \land z)) \land \neg \exists x' \exists y' ($

$((\neg x \land \neg y \land \neg x' \land y') \lor (\neg x \land y \land x' \land \neg y')$

$\lor (\neg x \land y \land x' \land y') \lor (x \land \neg y \land \neg x' \land \neg y') \lor (x \land y \land \neg x' \land \neg y')$

$)$

$\land \neg \exists x' \exists y' \exists z' ($

$((\neg x \land \neg y \land \neg z \land \neg x' \land y')$

$\lor (\neg x \land y \land \neg z \land x' \land \neg y' \land \neg z') \lor (\neg x \land y \land z \land x' \land y' \land \neg z')$

$\lor (x \land \neg y \land \neg z \land \neg x' \land \neg y' \land \neg z') \lor (x \land y \land \neg z \land \neg x' \land \neg y' \land \neg z')$

$) \land (\neg z' \lor (\neg x' \land y' \land z'))))$

$= (\neg z \lor (\neg x \land y \land z)) \land \neg \exists x' \exists y' (((\neg x \land \neg y \land z \land \neg x' \land y') \lor (\neg x \land y \land x' \land y')$

$\lor (x \land \neg y \land z \land \neg x' \land \neg y') \lor (x \land y \land z \land \neg x' \land \neg y')))$

$= (\neg z \lor (\neg x \land y \land z)) \land \neg((\neg x \land \neg y \land z) \lor (\neg x \land y) \lor (x \land \neg y \land z) \lor (x \land y \land z))$

Symbolic simulation-checking - an example

$B_1 = (\neg z \lor (\neg x \land y \land z)) \land \neg((\neg x \land \neg y \land z) \lor (\neg x \land y) \lor (x \land \neg y \land z) \lor (x \land y \land z))$

$= (\neg z \lor (\neg x \land y \land z)) \land \neg((\neg x \land \neg y \land z) \lor (\neg x \land y) \lor (x \land \neg y \land z) \lor (x \land y \land z))$

$= (\neg z \lor (\neg x \land y \land z)) \land \neg(z \lor (\neg x \land y \land \neg z))$

$= (\neg z \lor (\neg x \land y \land z)) \land \neg(z) \land \neg(\neg x \land y \land \neg z)$

$= (\neg z \lor (\neg x \land y \land z)) \land \neg(z) \land \neg(\neg x \land y \land \neg z)$

$= (\neg x \land \neg y \land \neg z) \lor (x \land \neg y \land \neg z) \lor (x \land y \land \neg z)$

Symbolic simulation-checking - an example

$B_2 = ((\neg x \land \neg y \land \neg z) \lor (x \land \neg y \land \neg z) \lor (x \land y \land \neg z)) \land \neg \exists x' \exists y' ($

$((\neg x \land \neg y \land \neg x' \land y') \lor (\neg x \land y \land x' \land \neg y')$

$\lor (\neg x \land y \land x' \land y') \lor (x \land \neg y \land \neg x' \land \neg y') \lor (x \land y \land \neg x' \land \neg y')$

$)$

$\land \neg \exists x' \exists y' \exists z' ($

$((\neg x \land \neg y \land \neg z \land \neg x' \land y')$

$\lor (\neg x \land y \land \neg z \land x' \land \neg y' \land \neg z') \lor (\neg x \land y \land z \land x' \land y' \land \neg z')$

$\lor (x \land \neg y \land \neg z \land \neg x' \land \neg y' \land \neg z') \lor (x \land y \land \neg z \land \neg x' \land \neg y' \land \neg z')$

$) \land ((\neg x' \land \neg y' \land \neg z') \lor (x' \land \neg y' \land \neg z') \lor (x' \land y' \land \neg z'))))$

$= ((\neg x \land \neg y \land \neg z) \lor (x \land \neg y \land \neg z) \lor (x \land y \land \neg z)) \land \neg \exists x' \exists y' ($

$((\neg x \land \neg y \land \neg x' \land y') \lor (x \land \neg y \land z \land \neg x' \land \neg y') \lor (x \land y \land z \land \neg x' \land \neg y')))$

$= ((\neg x \land \neg y \land \neg z) \lor (x \land \neg y \land \neg z) \lor (x \land y \land \neg z)) \land \neg((\neg x \land \neg y) \lor (x \land \neg y \land z) \lor (x \land y \land z)))$

Symbolic simulation-checking - an example

B_2

$= ((\neg x \land \neg y \land \neg z) \lor (x \land \neg y \land \neg z) \lor (x \land y \land \neg z)) \land \neg((\neg x \land \neg y) \lor (x \land \neg y \land z) \lor (x \land y \land z)))$

$= (x \land \neg y \land \neg z) \lor (x \land y \land \neg z)$

Here, the initial statepair has been elimianted.