State Machines
Formal Methods
Lecture 3

Farn Wang
Dept. of Electrical Engineering
National Taiwan University

Purpose

Understanding

- the formal semantics of programs
- the definition of state transition systems
- Problems of finite-state system analysis
- Algorithms for the problems

Organization

- Sequential Program – Operational Semantics
- Kripke Structures
- Concurrent Systems
- Verification problems of state-transition systems

State-space representations from programs - States, transitions

- Program Variables
  - Program Counter (pc), data variables, …
- Program State
  - Valuation of program variables
- Transition
  - Moving one state to another by executing a program statement.

Kripke structure from programs - operational Semantics

- Operational Semantics clarifies the execution of a program.
- Closes the gap between the text of a program and the behaviors represented by it.
- Let us look only at sequential programs for the moment.

IMP : a toy imperative language

- IMP is an imperative language in the style of PASCAL or C (even though some of the syntax may be different)
- The language contains arithmetic and boolean expressions as well as if-then-else, while statements.
- The syntax of the program will be described by BNF grammars.

IMP : a toy imperative language

- During execution of IMP program, the state of execution will be captured by the values of program variables.
- Operational semantics will be described by rules which specify how
  - Expressions in IMP pgm. are evaluated
  - Statements in IMP pgm. change the state

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.

$$A ::= c \mid x \mid (M) \mid A_1{+}A_2 \mid A_1{-}A_2$$
$$M ::= c \mid x \mid (A) \mid M_1{*}M_2 \mid M_1/M_2$$

c is an integer

x is a variable name.

BNF, syntax definitions
- Examples of context-sentivity

Session I:
- A: *Are you married ?*
- B: *No!*
- A: *Do you have children ?*
- B: ☹

Session 3:
- A: *Are you married ?*
- B: *Yes!*
- A: *Do you have children ?*
- B: ☺

Rude contextual interpretation: Are you a single parent ?

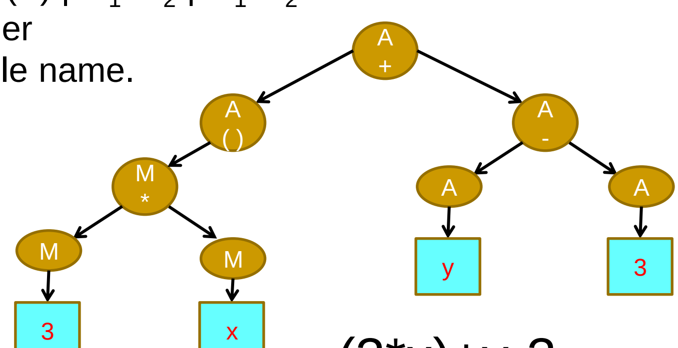
Session 2:
- A: *Do you have children ?*
- B: *Yes!*
- A: *Are you married ?*
- B: ☹

Session 4:
- A: *Do you have children ?*
- B: *No!*
- A: *Are you married ?*
- B: ☺

Rude contextual interpretation: Are you a single parent ?

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.

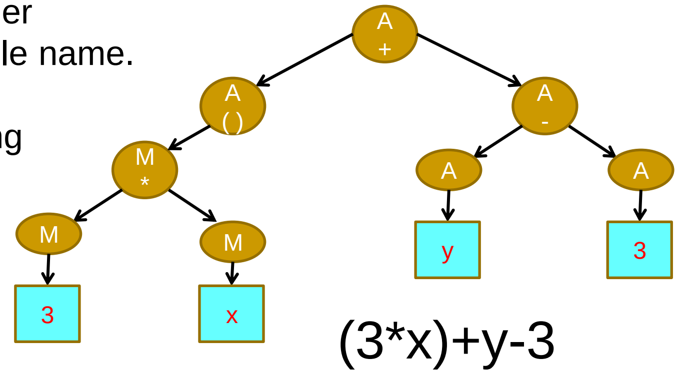


(3*x)+y-3

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

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

used in string generation.

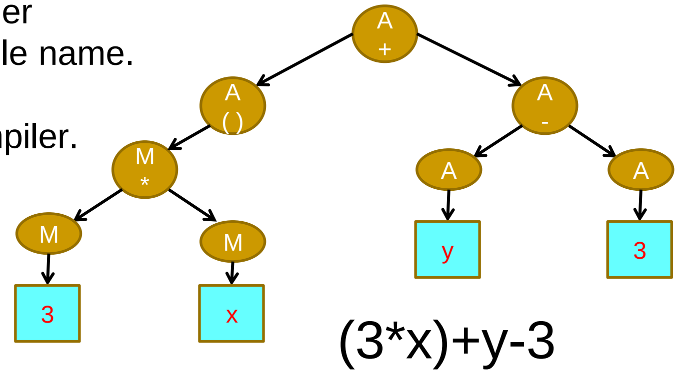


(3*x)+y-3

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

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

used in compiler.

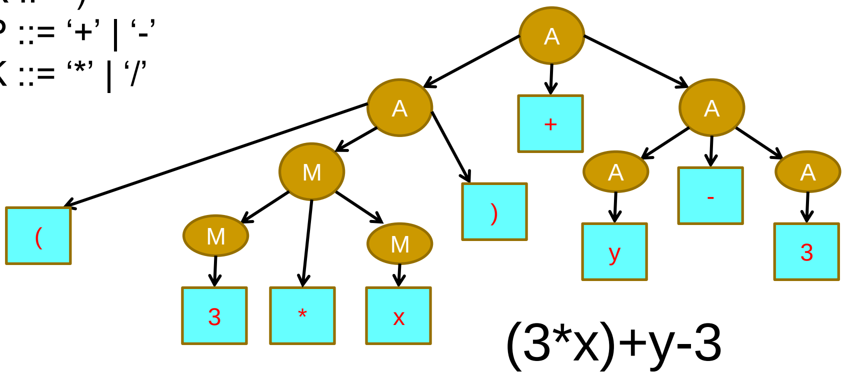


(3*x)+y-3

BNF, another syntax definition

A ::= c | x | L M R | A_1 P A_2
M ::= c | x | L A R | M_1 K M_2
L ::= '('
R ::= ')'
P ::= '+' | '-'
K ::= '*' | '/'

c is an integer
x is a variable name.



(3*x)+y-3

Syntax of IMP

- Non-negative integers *N*
- Truth values *T* = {true, false}
- Variables *V*
- Arithmetic expressions *A*
- Boolean expressions *B*
- Statements/commands *C*

Syntax of expression

<u>Arithmetic expressions</u>
$A ::= c \mid x \mid A_1 \oplus A_2 \mid (A) \mid (B)?A_1:A_2$

$c \in \aleph$, x is a variable.
$\oplus \in \{+,-,*,/,\%\}$

<u>Boolean expressions</u>
$B ::= \mathit{true} \mid A_1 \approx A_2 \mid {\sim}B_1 \mid B_1 || B_2 \mid (B_1)$

$\approx\ \in \{<=,<,==,!=,>,>=\}$
$\mathit{false} \equiv {\sim}\ \mathit{true}$, $B_1 => B_2 \equiv ({\sim}\ B_1) || B_2$,
$B_1 \&\& B_2 \equiv {\sim}(({\sim}B_1) || {\sim}B_2)$

Expressions - examples

- x + 2*y < 3
- x*y + y*y*3 == z
- x
- x + 2y < 3 || x*y + y*y*3 =< z
- (x + 2*y < 3 || ~x*y + y*y*3 == z)&&flag

Please construct the parsing trees.

Syntax of Commands *C*

$$
\begin{aligned}
C ::= &\ ; \\
&|\ x\text{=A;} \\
&|\ \{\ C_1\ \} \\
&|\ C_1 C_2 \\
&|\ \text{if (B) } C_1 \text{ else } C_2 \\
&|\ \text{while (B) } C_1
\end{aligned}
$$

IMP

- statement example

```
w = 0;
x = 0;
y = z*z;
while (x < y) {
  w = w + x*z;
  x = x + 1;
}
if (w > z*z*z) w = z*z*z;
```

Please construct the parsing tree.

Execution model

- Operational semantics of IMP describes how programs in that language are executed.
- To describe this, it needs to assume an underlying execution model.
- The execution model could be thought as a state machine although not necessarily a finite state machine.

Operational Semantics

Operational Semantics for the IMP language will give rules to describe the following:

Give a state *s*

- How to evaluate arithmetic expressions
- How to evaluate Boolean expressions
- How the commands can alter *s* to a new state *s'*

States

A state is a valuation of program variables i.e. each variable is mapped to a value in its type

- Thus, if {a,b} are the only variables in an IMP program, then each of the following are states in the execution model
  - a=0, b=0
  - a=0, b=1
  - a=0, b=2
  - …
  - a=1, b=0
  - …

Meaning of Arith. Expressions (1)

$\langle$A,s$\rangle$

- Numbers:$\langle$c,s$\rangle$= c

  Number *c* in any state *s* evaluates to *c*

  E.g. $\langle$0,s$\rangle$= 0, $\langle$5,s$\rangle$= 5
- Variables: $\langle x$,s$\rangle$= s(x) $\langle X$,s$\rangle\equiv$s(X)

  Variable *X* in state *s* evaluates to value of x in *s*.

  E.g. $\langle a,(a=5,b=20)\rangle = 5$, $\langle b,(a=5, b=20)\rangle = 20$

Meaning of Arith. Expressions (2)

$\langle A,s\rangle$

- Sums: $\langle a + b, s\rangle = \langle a, s\rangle + \langle b, s\rangle$
  e.g. $\langle a + b, (a = 5, b = 20)\rangle = 25$

- Products: $\langle a * b, s\rangle = \langle a, s\rangle * \langle b, s\rangle$
  e.g. $\langle a * b, (a = 5, b = 20)\rangle = 100$

23

Example arith. expr. evaluation

Evaluating meaning of a complicated arithmetic expression will require

- Several application of the above rules
- Operator precedence

EX: $\langle a * b + b, (a = 5, b = 20)\rangle$

$= \langle a * b, (a = 5, b = 20)\rangle + \langle b, (a = 5, b = 20)\rangle$

$= \langle a, (a = 5, b = 20)\rangle * \langle b, (a = 5, b = 20)\rangle + 20$

$= 5 * 20 + 20 = 120$

24

Meaning of Boolean Expression (1)

$\langle B, \mathrm{s} \rangle$

- $\langle \mathit{true}, \mathrm{s} \rangle = \mathit{true}$
- $\langle \mathit{false}, \mathrm{s} \rangle = \mathit{false}$
- Inequality Check:
  $\langle A_1 \approx A_2, \mathrm{s} \rangle = \langle A_1, \mathrm{s} \rangle \approx \langle A_2, \mathrm{s} \rangle$
- Negation:
  $\langle \sim B, \mathrm{s} \rangle = \sim\langle B, \mathrm{s} \rangle$
- Disjunction:
  $\langle B_1 \,||\, B_2, \mathrm{s} \rangle = \langle B_1, \mathrm{s} \rangle || \langle B_2, \mathrm{s} \rangle$

Workout

State s:(a=5, b=6)

- $\langle a=b, \mathrm{s}\rangle \equiv \langle a, \mathrm{s}\rangle = \langle b, \mathrm{s}\rangle \equiv 5=6 \equiv \mathit{false}$
- $\langle \sim a=b, \mathrm{s}\rangle = \sim\langle a=b, \mathrm{s}\rangle = \mathit{true}$
- $\langle a<=b, \mathrm{s}\rangle \equiv \langle a, \mathrm{s}\rangle <= \langle b, \mathrm{s}\rangle \equiv 5<=6 \equiv \mathit{true}$
- $\langle a<=b \,\&\&\, a=b, \mathrm{s}\rangle$
  $= \langle a<=b, \mathrm{s}\rangle \,\&\&\, \langle a=b, \mathrm{s}\rangle$
  $= \mathit{true} \,\&\&\, \mathit{false}$
  $= \mathit{false}$

Meaning of Expressions

- Expressions evaluate to values in a given state
- Therefore, the meaning of expressions are given by values.
  - Boolean values for boolean expressions
  - Number for arithmetic expressions
- Using the meaning of expressions, we can assign meaning to commands.

Workout

State s:(a=3, b=10, c=5)

1. ⟨a+3*b*c,s⟩=
2. ⟨a+3*b=c,s⟩=
3. ⟨¬ ,s⟩= *false*
4. ⟨ ≠ ,s⟩= *true*
5. ⟨ ∧ ,s⟩= *true*
6. ⟨ ∨ ,s⟩= *false*
7. ⟨ ≤ ,s⟩= *false*
8. ⟨ > ,s⟩= *false*
9. ⟨ ∧¬ ,s⟩= *true*
10. ⟨ ≠ ∧¬ ,s⟩= *true*
11. ⟨ → ,s⟩ = *false*
12. ⟨ ∨¬ ,s⟩= *true*
13. ⟨ →¬ ,s⟩= *true*

Meaning of Commands

- Execution of commands leads to a change of program state.
- Therefore the meaning of a command C is : If C is executed in some state s, how does it change s to s'.

$$\langle C, \mathrm{s}\rangle = \mathrm{s'}$$

Rules for commands (1)

$\langle$C, s$\rangle$

- $\langle$;, s$\rangle$ = s
- $\langle x$=A;, s$\rangle$ = s[x=A]
  - s[x=A] is the same as state s except that the value of x is $\langle$A, s$\rangle$.
  - Ex: $(a=5,b=20,c=2)[a=7] = (a=7,b=20,c=2)$
  - Ex: $(a=5,b=20,c=2)[a=5] = (a=5,b=20,c=2)$
  - Ex: $(a=5,b=20,c=2)[a=\mathrm{b}+\mathrm{c}] = (a=22,b=20,c=2)$
- $\langle\{ C_1 \}, s\rangle = \langle C_1, s\rangle$

Rules for commands (1)

$\langle C, s\rangle$

- $\langle\, C_1C_2, s\rangle \;=\; \langle\, C_2, \langle\, C_1, s\rangle\, \rangle$

- $\langle$ if (B) C_1 else C_2 , s$\rangle = \langle\, C_1, s\rangle$ if $\langle B, s\rangle$ = *true*
  $\langle$ if (B) C_1 else C_2 , s$\rangle = \langle\, C_2, s\rangle$ if $\langle B, s\rangle$ = *false*

- $\langle$while(B)C_1, s$\rangle$ = s if $\langle B,s\rangle$ = *false*
  $\langle$while(B)C_1, s$\rangle$ = $\langle$while(B)C_1, $\langle C_1,s\rangle\rangle$ if $\langle B,s\rangle$ = *true*

Summary of rules

- The meaning of each commands specifies how an execution of the command changes state.
- Roughly speaking, this is done by simulating the execution of the commands.
- For example, the rule for while essentially unfolds the iterations of while loop.

Kripke Structure

- A state-transition system that captures
  - What is true of a state
  - What can be viewed as an atomic move
  - The succession of states
- Static representation that can be unrolled to a tree of execution traces, on which temporal properties are verified

33

Kripke st

I'm honored by your proposal, by my mum says I have to finish high-school first.

Saul Kripke

Bo

Pr

Di

-
- ... of languages
- ...tgenstein
-

- wrote his first essay on Kripke structure at 16
- invited to teach at Princeton
- taught a graduate logic course at MIT since sophomore year at Harvard.

34

Kripke structure
- syntax

To extend to integer programs, $L: S \times P \rightarrow \mathbb{N}$

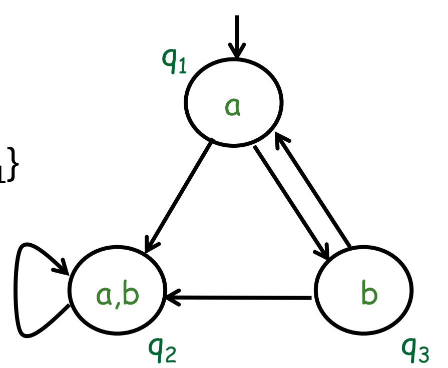
$A = (S, S_0, R, L)$

- 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

- *L* allows us to describe the truth/falsehood of a proposition in the various states of a system.
- The propositions refer to valuations of the state variables.

Kripke Model
- syntax

- Set of states $S=\{q_1,q_2,q_3\}$
- Set of initial states $S_0=\{q_1\}$
- $R =\{(q_1,q_2),(q_2,q_2),$
  $(q_1,q_3),(q_3, q_1),$
  $(q_3,q_2)\}$



- Set of atomic propositions $AP=\{a,b\}$
- $L(q_1)=\{a\}$, $L(q_2)=\{a,b\}$, $L(q_3)=\{b\}$

Kripke structure
- semantics

Given a Kripke structure $A = (S, S_0, R, L)$,

a run is a finite or infinite sequence

$$s_0 s_1 s_2 \ldots s_k \ldots$$

such that

- $s_0 \in S_0$
- for each $k \in \mathbb{N}$, if s_{k+1} exists,
  - $s_{k+1} \in S$ and
  - $R(s_k, s_{k+1})$ is true.

Control and data variables

- State = valuation of control and data vars.
- In our example
  - *pc0, pc1* are control variables.
  - *turn* is a shared data variable.
- To generate a finite state transition system
  - Data variables must have finite types, and
  - Finitely many control locations

Program → Kripke structure
- Data variables

Data variables often do not have finite types

- integer, …
- Usually abstracted into a finite type.
- An integer variable can be abstracted to {-,0,+}
- Just store the information about the sign of the variable. (coming up with these abstractions is a whole new problem).

39

Program → Kripke structure
- Control Locations

Isn't the control locations of a program always finite ?

- NO, because your program may be a concurrent program with unboundedly many processes or threads (parameterized system).
- Can employ control abstractions (such as symmetry reduction)

40

2009/10/28 stopped here.

Program → Kripke structure - States and Transitions

- Each component makes a move at every step.
- Digital circuits are most often *synchronous*.
  - Common clock driving the system.
  - Contents of flip-flops define the states.
  - On every clock pulse, the content of every flip-flop (potentially) changes.
- This change is captured by the transition relation.

Program → Kripke structure
- States and Transitions

- Define $V=\{v_1,\ldots,v_n\}$, boolean variables representing state of flip-flops in the circuit.
- Set of states represented by boolean formula over $v_1,\ldots,v_n$.
- To define transitions, define a fresh set of variables $V'=\{v'_1,\ldots,v'_n\}$. These are the next state variables.
- The transitions are now represented by a relation $R(V,V')\subseteq V\times V'$

43

Kripke structure
- Transition Relation

- $(s,s')\in R(V,V')$ implies $s \rightarrow s'$
- Now, $R(V,V')=\cup_{i\in\{1,\ldots,n\}} R_i(V,V')$, where captures the changes in state variable v_i
- Define $R_i(V,V') =(v'_i\Leftrightarrow f_i(V))$ where $f_i(V)$ is a boolean function defining the value of flip-flop *i* in next state.
- Given a synchronous circuit, we then need to define $f_i(V)$ for each *i*.

44

Transition relation - A synchronous mod 8 counter

- $V=\{v_2,v_1,v_0\}$, where v_0 is the least significant bit.
- The transitions can be enumerated as:

  000→001→010→……

- Alternatively define how each of the three bits are changed on every clock cycle
  - $v'_0 = \neg v_0$ (the least significant bit)
  - $v'_1=v_0 \oplus v_1$
  - $v'_2=(v_0 \wedge v1) \oplus v_2$ (the most significant bit)

Kripke Structure - example

Suppose there is a program

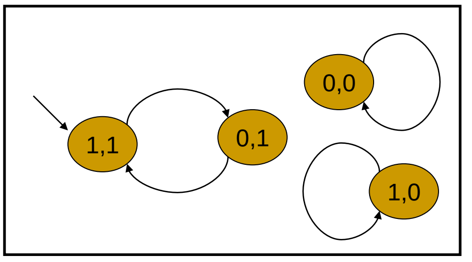
```
initially x==1 && y==1;
while (true)
  x = (x+y) % 2;
```

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

Kripke Structure
- example

Suppose there is a program

```
initially x==1 && y==1;
while (true)
  x = (x+y) % 2;
```



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

Kripke Structure
- example

Suppose there is a program

```
initially x==1 && y==1;
while (true)
  x = (x+y) % 2;
```

S=DxD ={(0,0),(0,1),(1,0),(1,1)}
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}

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

Kripke Structure
- example

Suppose there is a program

```
initially x==1 && y==1;
while (true)
  x = (x+y) % 2;
```

S=*D*x*D* ={a,b,c,d}
S_0={a}
R={(a,b),(b,a),
(c,c),(d,d)}
L(a)={x=1,y=1},
L(b)={x=0,y=1},
L(c)={x=1,y=0},
L(d)={x=0,y=0}

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

Workout
- Kripke Structure

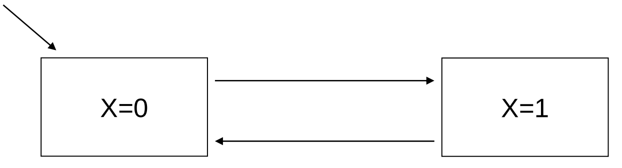
Suppose there is a program

```
initially x==1 && y==1;
while (true)
  x = (x+y) % 3;
```

where x and y range over *D*=[0,2]

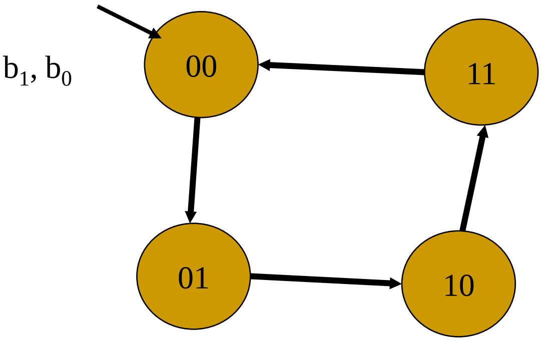
Kripke Structure
- an example

Initially x=0
While (true)
  x:=1-x;



Kripke Structure
- example

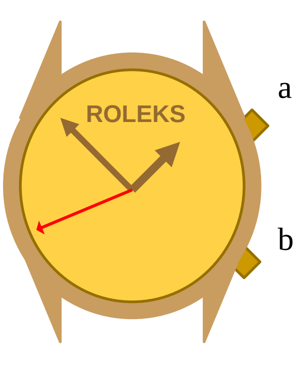
A 2-bit counter operates at bit-level.

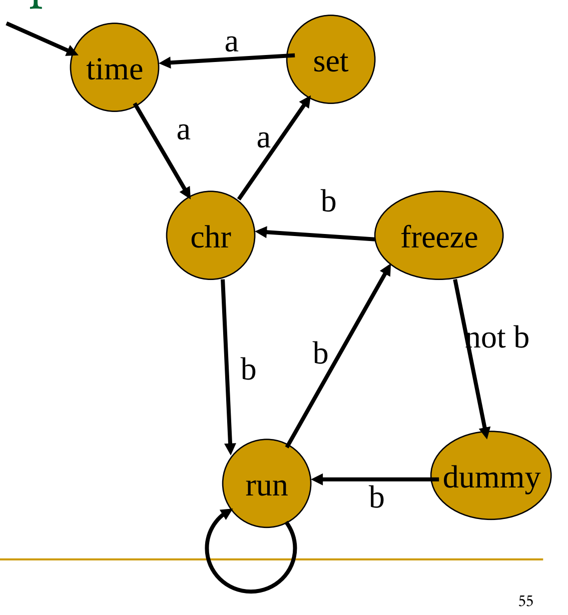


Kripke Structure
- workout

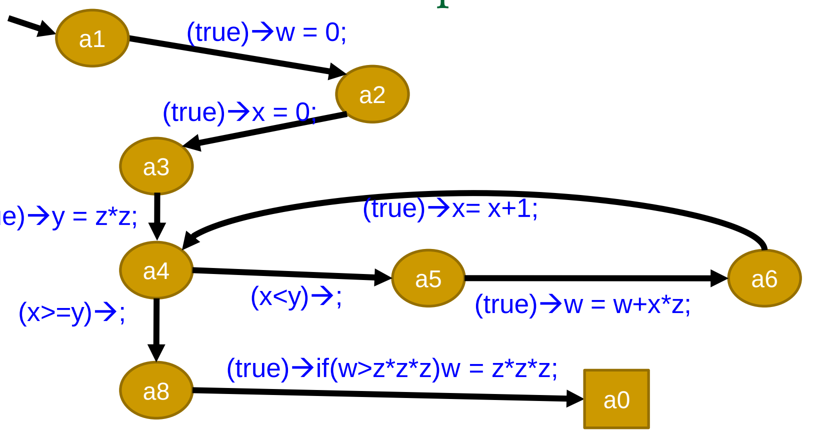
Write a simple program for the Kripke structures in the last page.

Automata & Kripke structure

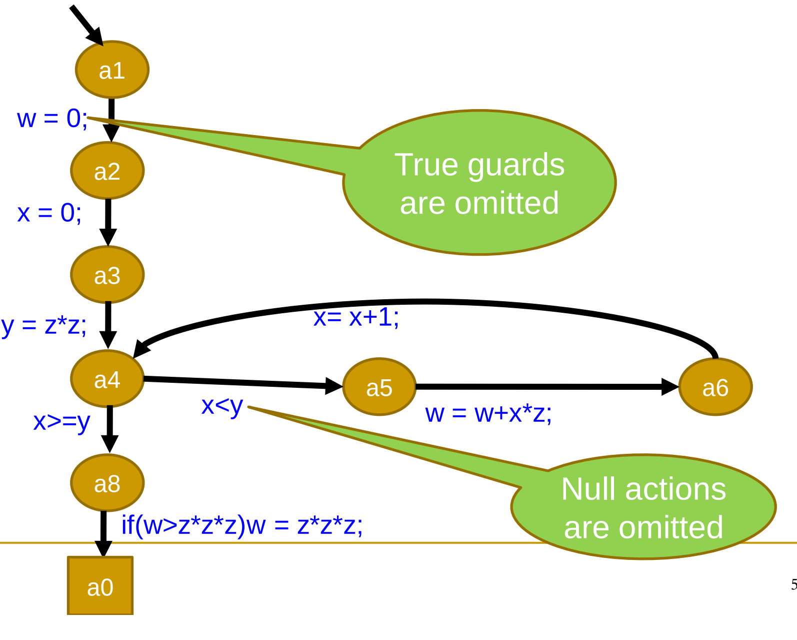




State-transition graphs
- an extension of automata
for complex models



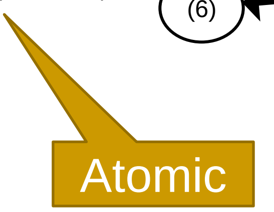
State-transition graphs

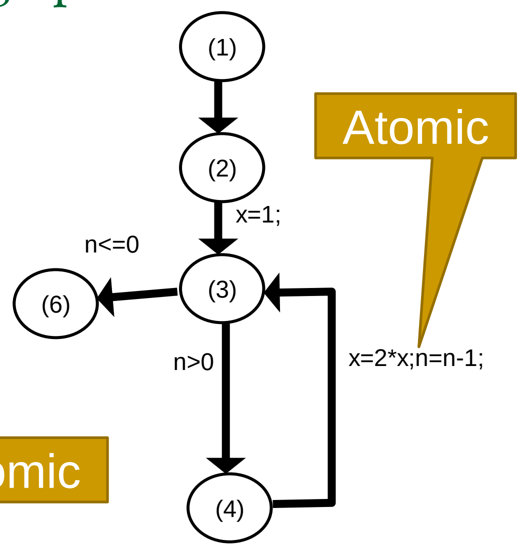


State transition graphs

- from a program

(1) f(n) {
(2)   x =1;
(3)   while (n > 0) {
(4)   x = x*2; n=n-1;
(5)   }
(6)   return x;
(7) }

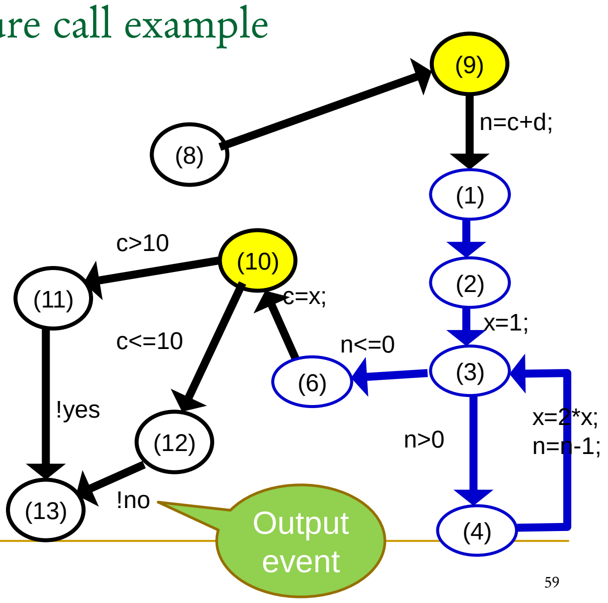




State-transition graphs

- from a procedure call example

(1)   f(n) {
(2)   x =1;
(3)   while n > 0, do {
(4)   x = x*2; n=n-1;
(5)   }
(6)   return x;
(7)   }
(8)   main (c,d) {
(9)   c = f(c+d);
(10)   if (c > 10)
(11)   print “yes”.
(12)   else print “no”.
(13) }



Guarded commands with modes (GCM)

- A text language for state-transition graphs
- For multi-thread systems
- Extension with programming concepts

Process count declaration
Variable declaration
Inline expression declaration (optional)
Mode declaration
Initial condition
Specification (optional)

Guarded commands with modes (GCM) - a language for state-transition graphs

V is a variable declaration.

E is an arithmetic expression.

B is a Boolean condition.

C is a program of IMP commands or "goto name" where name is a mode name.

Each rule R is executed atomically.

- for the modeling of complex behaviors in transitions.

A program can be a set of GCM.

- At any moment, at most one command is executed.

Guarded commands with modes (GCM) - a language for state-transition graphs

G ::= P VS [ILS] MS INI [SP]

P ::= *process count =* E*;*

VS ::= | V VS

V ::= SCOPE TYPE x [*:* c *..* c]*;* // $c \in$ N, x a variable

SCOPE ::= *global* | *local*

TYPE ::= *discrete* | *pointer* | *clock* | *dense*

| *synchronizer*

Threads are indexed 1 through c.

Guarded commands with modes (GCM) - a language for state-transition graphs

IL ::= *inline* TYPE *name* *(* FSL *)* *{* EI *}*

FSL ::= | FS

FS ::= f | f *,* FS // f : a formal argument

EI ::= f | x | x*[c]* // x : a declared discrete variable

| *(* EI *)* | EI*+*EI | EI*-*EI | EI***EI | EI*/*EI | EI*%*EI

| *(* BI *)* *?* EI *:* EI | *#PS* | *P*

| *name* *(* EISS *)*

EISS ::= | EIS

EIS ::= EI | EI *,* EIS

Guarded commands with modes (GCM) - a language for state-transition graphs

BI ::= *(* BI *)* | EI<=EI | EI<EI | EI>=EI | EI>EI
| EI==EI | EI!=EI
| BI *&&* BI | BI || BI | ~BI | BI => BI
| *forall* *x : c .. c,* BI | *exists* *x : c .. c,* BI
| *name* *(* EISS *)*

Guarded commands with modes (GCM) - a language for state-transition graphs

MS ::= | M MS
M ::= [*urgent*] *mode* *name* *(*B*)* *{* RS *}*
B ::= *(*B*)* | E<=E | E<E | E>=E | E>E | E==E | E!=E
| B *&&* B | B || B | ~B | B=>B E | *name* *(* ESS *)*
| *forall* *x : c .. c,* B | *exists* *x : c .. c,* B
E ::= *x* | *x[c]* // *x* : a declared discrete variable
| *(*E*)* | E+E | E-E | E*E | E/E | E%E
| *(*B*)*?E *:* E | *name* *(* ESS *)*
ESS ::= | ES
ES ::= E | E , ES

Guarded commands with modes (GCM) - a language for state-transition graphs

RS ::= | R RS

R ::= *when* SS *(*B*)* *may* C

SS ::= | S SS

S ::= *?x* | *?(*E*)x* | *!x* | *!(*E*)x* // *x* is a global synchronizer

| *?x@q* | *?x@(*E*)* | *!x@q* | *!x@(*E*)*

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

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

Guarded commands with modes (GCM) - a language for state-transition graphs

model threads

INI ::= *initially* B*;*

SP ::= RTASK B*;* | *tctl* T*;* | GTASK GS *;* GS *;*

spec threads

RTASK ::= *safety* | *goal* | *risk*

T ::= B | *(*T*)* | *forall always* K T | *exists always* K T

| *forall eventually* K T | *exists eventually* K T

| *forall* T *until* K T | *exists* T until K T

| *forall x : c .. c,* T | *exists x : c .. c,* T

K ::= *{[c , c]}* | *{(c , c]}* | *{[c ,* D*)}* | *{(c ,* D*)}*

D ::= *c* | *oo*

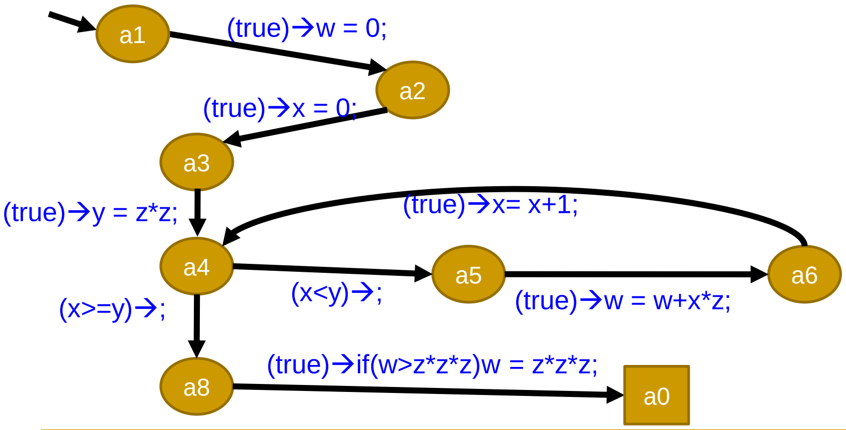
Guarded commands with modes (GCM) - a language for state-transition graphs

GTASK ::= check branching simulation
| check branching bisimulation

GS ::= c | c , GS

a sequence of thread indices for a particular roles

A state-transition - represented as a GCM



A state-transition
- represented as a GCM

```
pocess count = 1;
global discrete w,x,y,z:0..5;
mode a1 (true) { when (true) may w = 0; goto a2; }
mode a2 (true) { when (true) may x = 0; goto a3; }
mode a3 (true) { when (true) may y = z*z; goto a4; }
mode a4 (true) { when (x>=y) may goto a8;
                 when (x < y) may goto a5; }
mode a5 (true) { when (true) may w=w+x*z; goto a6; }
mode a6 (true) { when (true) may x=x+1; goto a4; }
mode a8 (true) { when (true) may if (w>z*z*z) w= z*z*z; }
initially a1[1]&&w==1&&x==1&&y==1&&z==1;
```


A state-transition
- represented as a GCM

```
pocess count = 1;
global discrete w,x,y,z:0..5;
mode a1.2 (true) { when (true) may w = 0; x = 0; goto a3; }
mode a3 (true) { when (true) may y = z*z; goto a4; }
mode a4 (true) { when (x>=y) may goto a8;
                 when (x < y) may goto a5; }
mode a5 (true) { when (true) may w=w+x*z; goto a6; }
mode a6 (true) { when (true) may x=x+1; goto a4; }
mode a8 (true) { when (true) may if (w>z*z*z) w= z*z*z; }
initially a1[1]&&w==1&&x==1&&y==1&&z==1;
```

Guarded commands with modes (GCM)

```
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;
```

program

guarded commands

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


Concurrent programs

- A set programs running independently, communicating from time to time, thereby performing a common task.
- *Flavors of Concurrency*
  - Synchronous execution
  - Asynchronous / interleaved execution
    - Communication via shared variables
    - Message passing communication

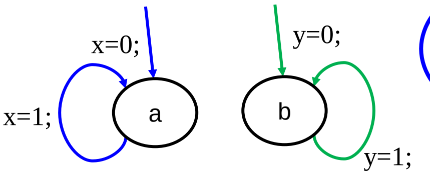
Kripke Structure
- for a concurrent system

- Programs (as opposed to circuits) are typically considered asynchronous.
- An asynchronous concurrent system is a collection of sequential programs $P_1 \dots P_k$ running in parallel with only one pgm. making a move at every time step.
  - How do the sequential programs communicate ?
  - What are the behaviors of the concurrent system ?

Kripke Structure
- for a concurrent system

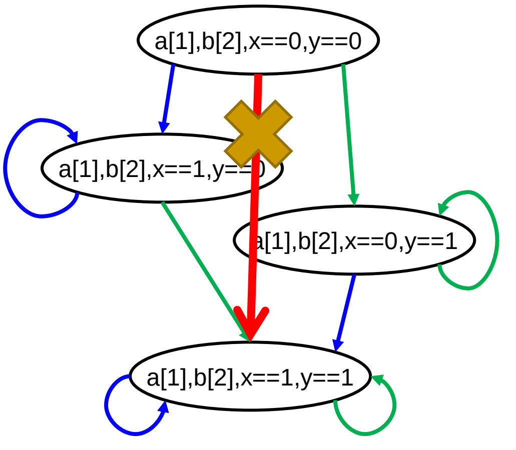
- Behaviors of each sequential program P_i captured by its operational semantic.
- The programs P_i need not be terminating.
- Behaviors (Traces) of $P_1 \dots P_k$ formed by interleaving the transitions of the programs.
- Consider two non-communicating programs.

Guarded commands
- for a concurrent system
Interleavings



state-transition graphs

Semantics as Kripke structure



Guarded commands
- for a concurrent system
Interleavings

process count = 2;

global discrete x, y:0..1;

mode a (true) {

when (true) may x=1; }

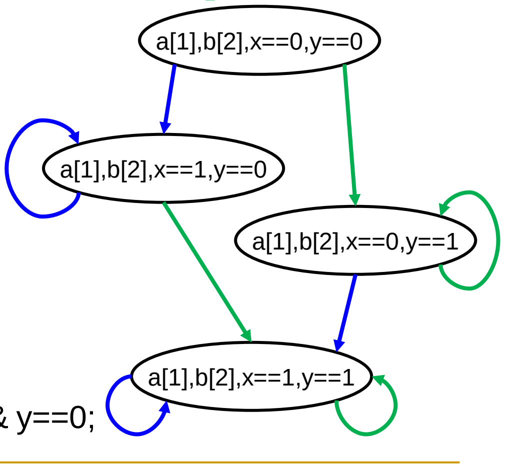
mode b (true) {

when (true) may y=1; }

initially a[1]&&b[2]&&x==0&& y==0;

GCM

Semantics as Kripke structure



2009/11/04 stopped here.

Kripke Structure
- for a concurrent system

- Obtaining Kripke Structure from a concurrent program directly is laborious.
- Typically, model checking tools allow you to input the program in its modeling language, and then it extracts the Kripke Structure (or some succinct version of it).
- Model the sequential pgms. separately and specify a model of concurrency

  e.g. asynchronous with shared variable communication

Kripke Structure
- A Mutual Exclusion Example

```
// 2 processes that communicate with a shared variable.
process count = 2;
global discrete turn: 0..1;

// state-transition graph for process 1
mode a0 (true) { when (turn==0) may goto a1; }
mode a1 (true) { when (true) may turn = 1; goto a0; }

// state-transition graph for process 2
mode b0 (true) { when (turn==1) may goto b1; }
mode b1 (true) { when (true) may turn = 0; goto b0; }

initially a0[1] && b0[2];
```


Kripke Structure
- for a concurrent system states

states can be recorded as

(mode of 1, mode of 2, value of turn)

- mode of 1 $\in$ {a0, a1}
- mode of 2 $\in$ {b0, b1}
- The value of turn $\in$ {0, 1}
- There are 8 states.
- Not all of them are reachable from the initial state.

09/11/18 stopped here.

State-transition graphs

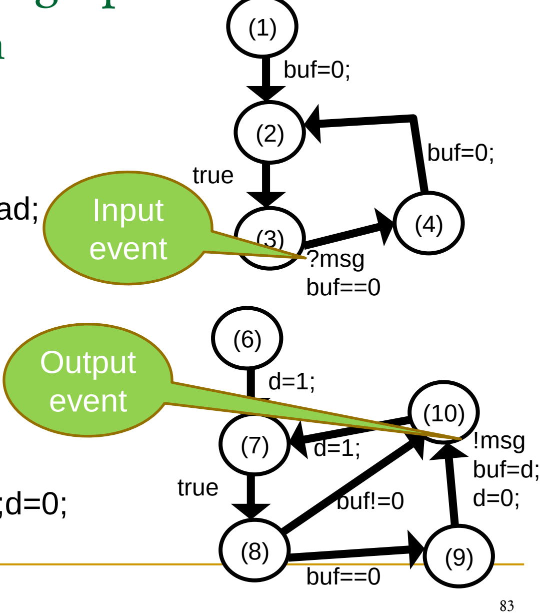
- Synchronization

The reader process

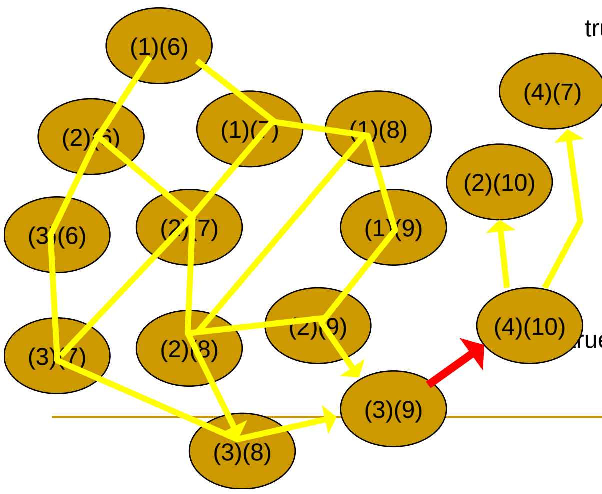
(1) buf = 0;
(2) while true, do {
(3) if (buf == 0), read;
(4) buf = 0;
(5) }

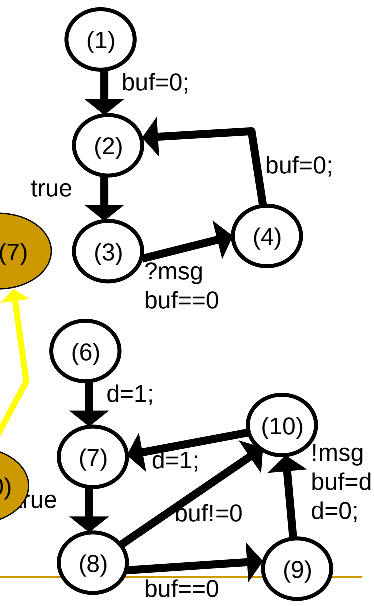
The writer process

(6) d = 1;
(7) while true, do {
(8) if (buf == 0),
(9) write buf = d;d=0;
(10) d = 1;
(11) }



State-transition graphs
- Synchronization
Kripke structure (part)

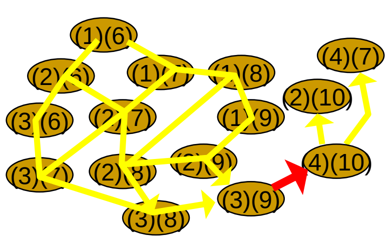


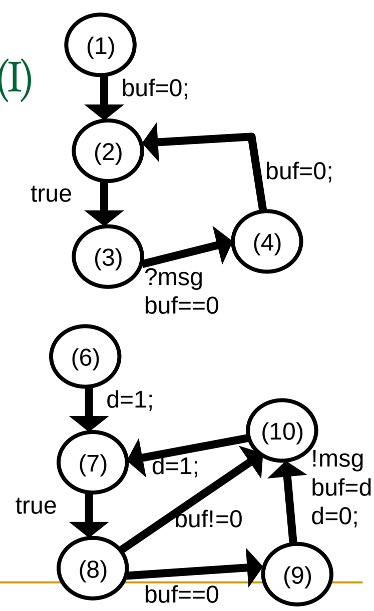


State-transition graphs
- Semantics of concurrency (I)

Interleaving semantics
(the yellow arcs).

- At most one autonomous party may execute at a time.





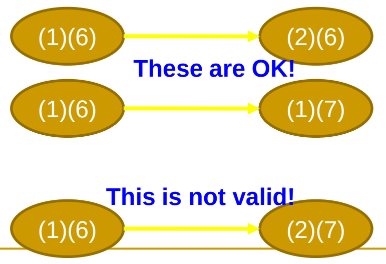
State-transition graphs

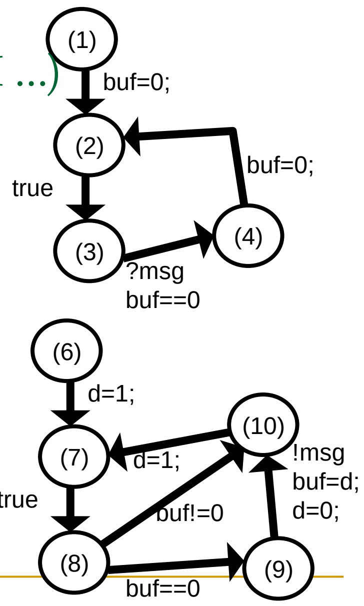
- Semantics of concurrency (I …)

Interleaving semantics
(the yellow arcs).

- At most one autonomous party may execute at a time.

E.g.,





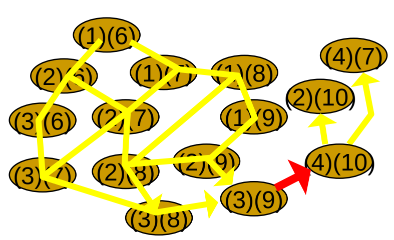
86

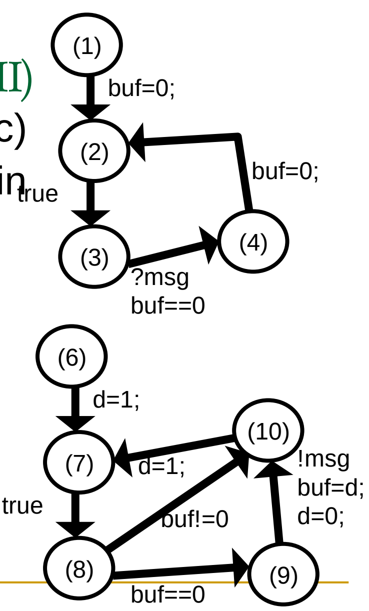
State-transition graphs

- Semantics of concurrency (II)

Synchronizations (the red arc)

- All communicating parties in a minimal & autonomous synchronization must execute at the same time.





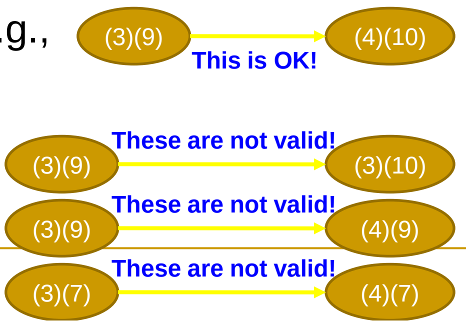
87

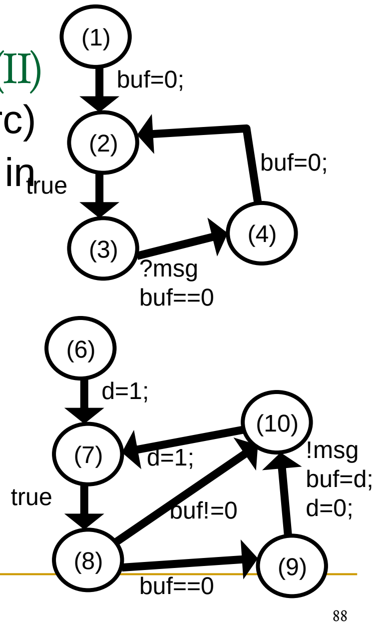
State-transition graphs
- Semantics of concurrency (II)

Synchronizations (the red arc)

- All communicating parties in a minimal & autonomous synchronization must execute at the same time.

E.g.,



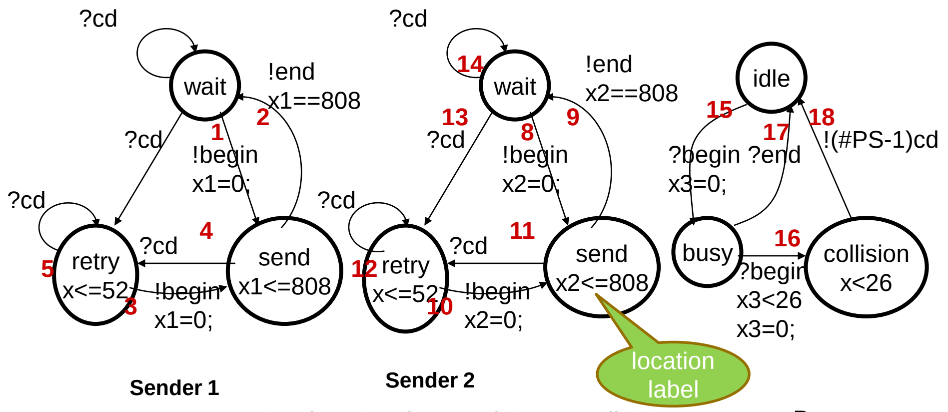


State-transition graphs

CSMA/CD protocol, the Ethernet protocol

- 500m in expanse
- 2500m in expanse with repeaters
  - Round-trip 48 μs.
- Messages length at least 64 bytes to detect round-trip corruption.

State-transition graphs
- CSMA/CD



Sender1,Sender2, and Bus are all PTAs.

State-transition graph for automata - an exercise

Please construct an automata with

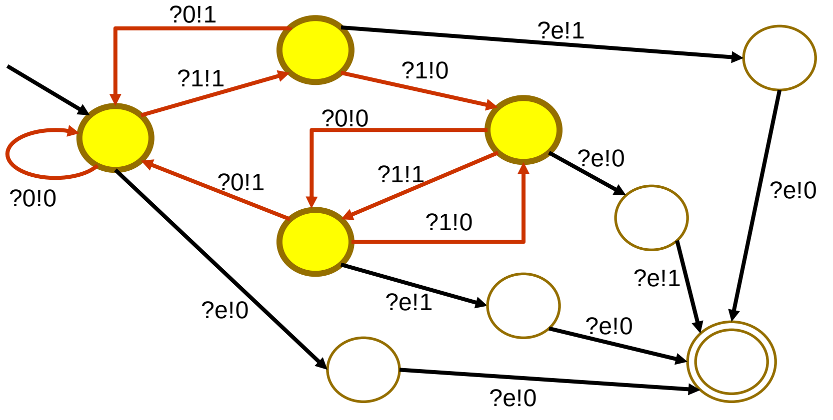
- input alphabet {1,0,e}
- output alphabet {1,0}
- reads in $eeb_n b_{n-1} \ldots b_1 b_0$
- output $3*(b_n b_{n-1} \ldots b_1 b_0)$ with b_n as the most significant bit.

Example:

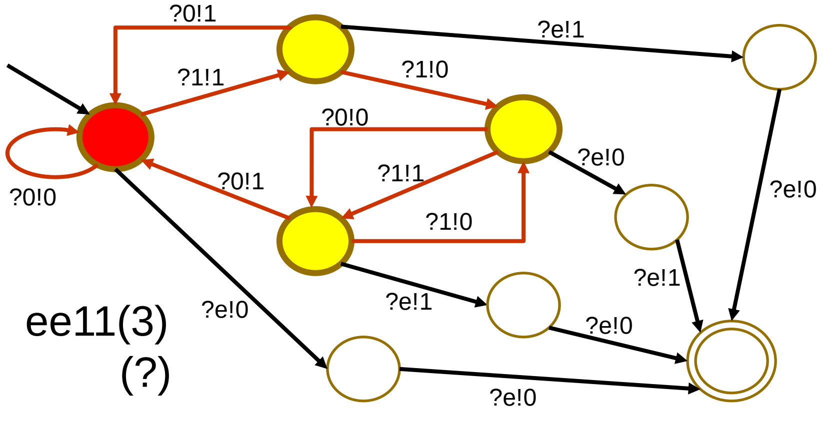
when input is ee1011(11), output is 100001(33)

ee11(3), 1001(9)

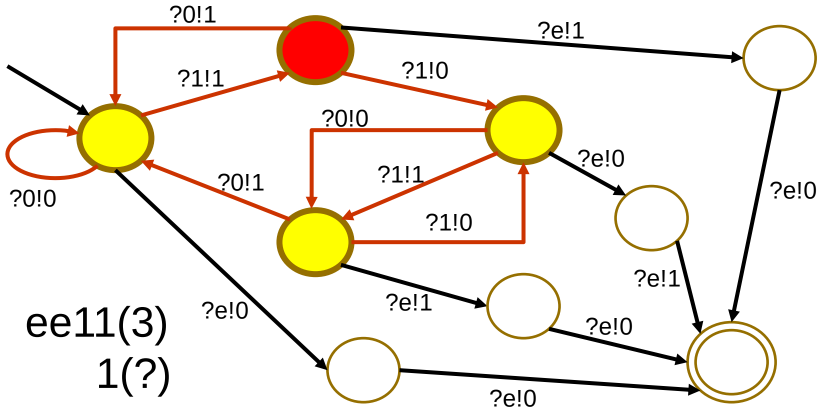
State-transition graph for automata - an exercise



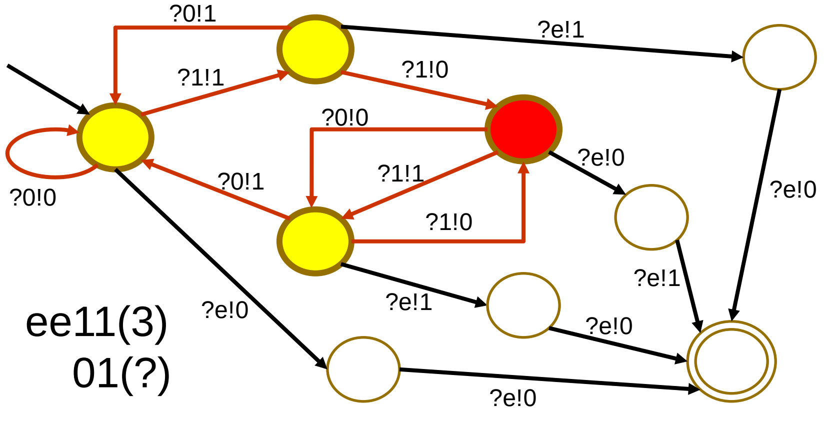
State-transition graph for automata - an exercise run for ee11(3)



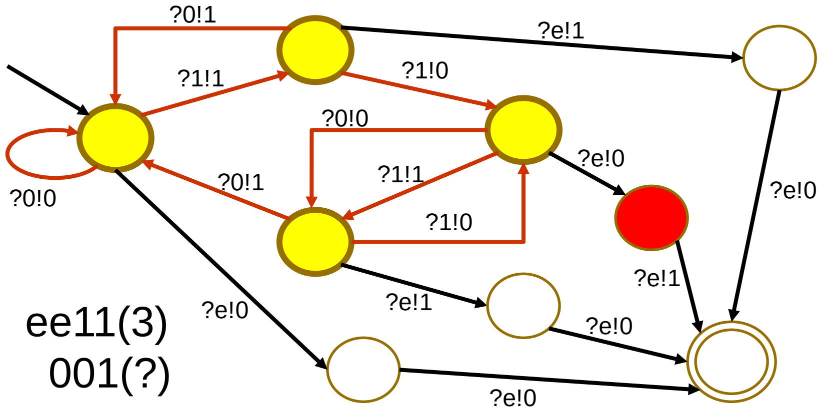
State-transition graph for automata - an exercise run for ee11(3)



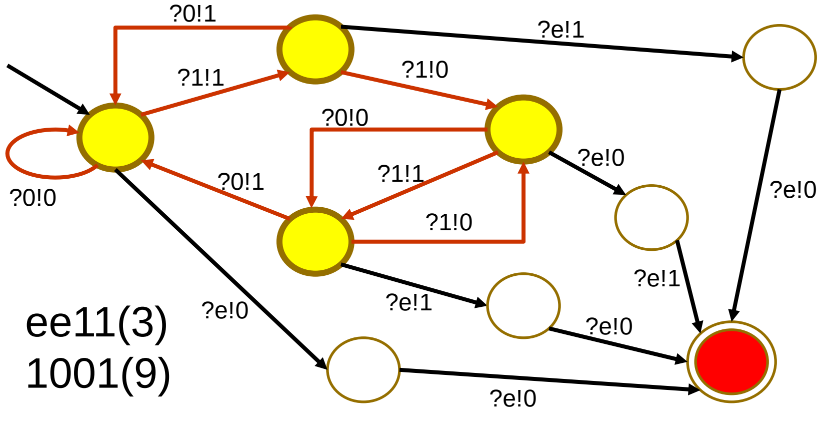
State-transition graph for automata - an exercise run for ee11(3)



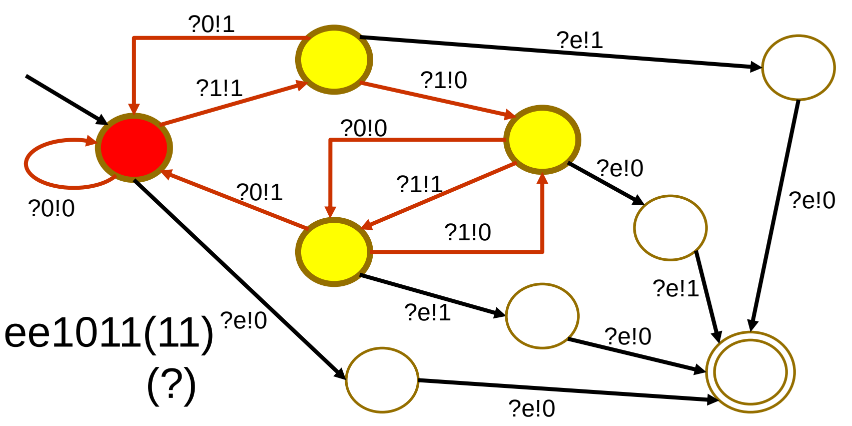
State-transition graph for automata - an exercise run for ee11(3)



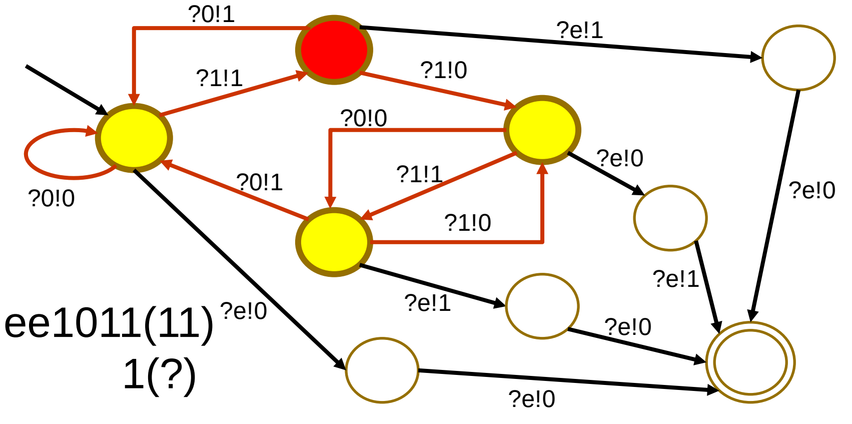
State-transition graph for automata - an exercise run for ee11(3)



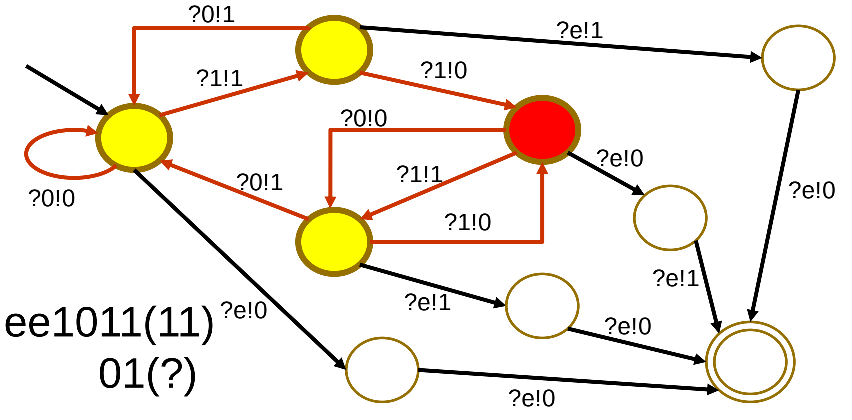
State-transition graph for automata - an exercise run for ee1011(11)



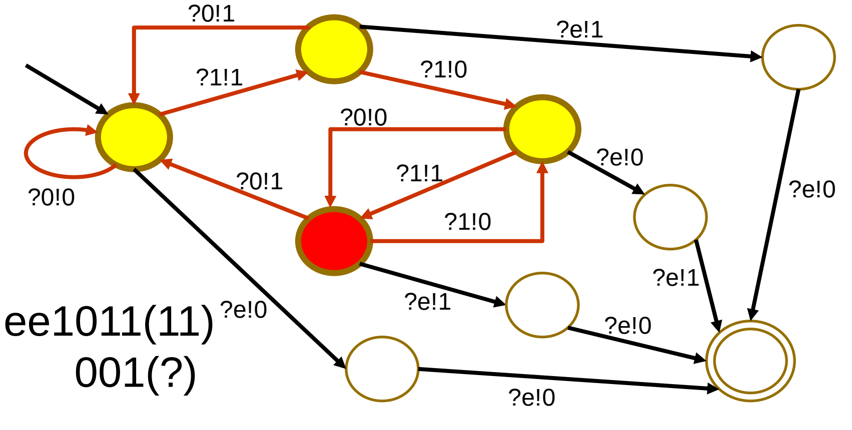
State-transition graph for automata - an exercise run for ee1011(11)



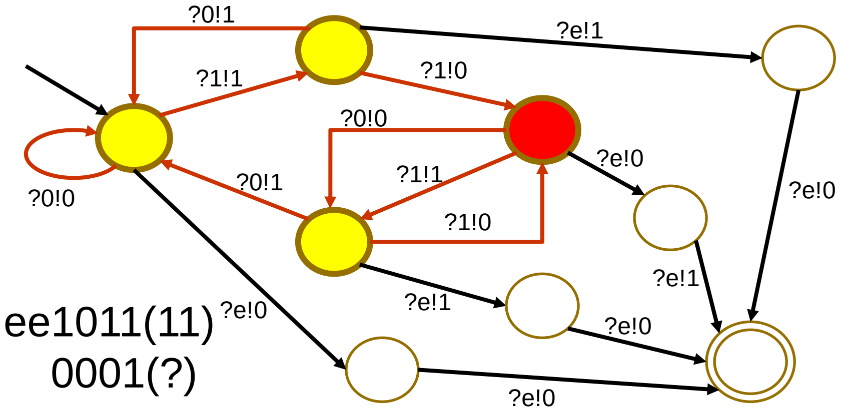
State-transition graph for automata - an exercise run for ee1011(11)



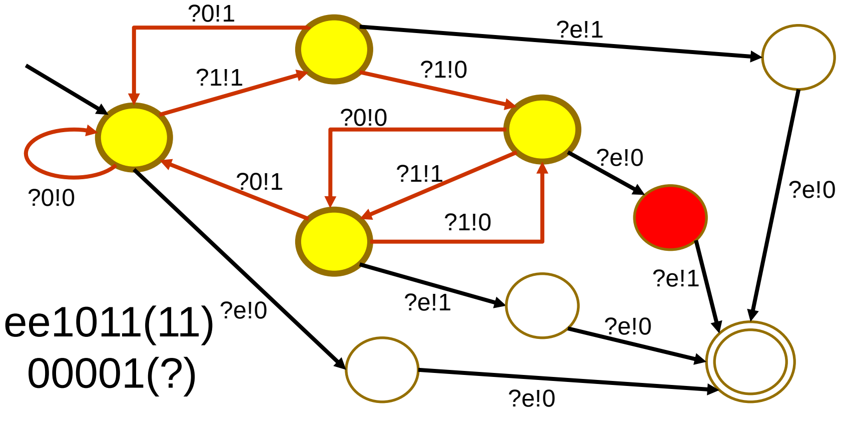
State-transition graph for automata - an exercise run for ee1011(11)



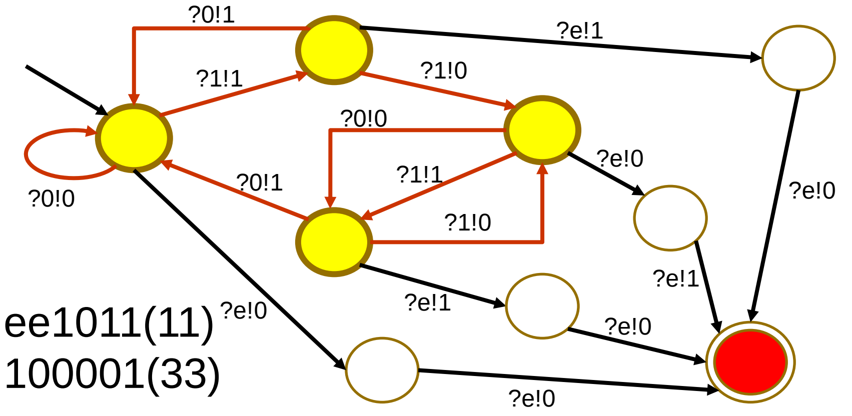
State-transition graph for automata - an exercise run for ee1011(11)



State-transition graph for automata - an exercise run for ee1011(11)

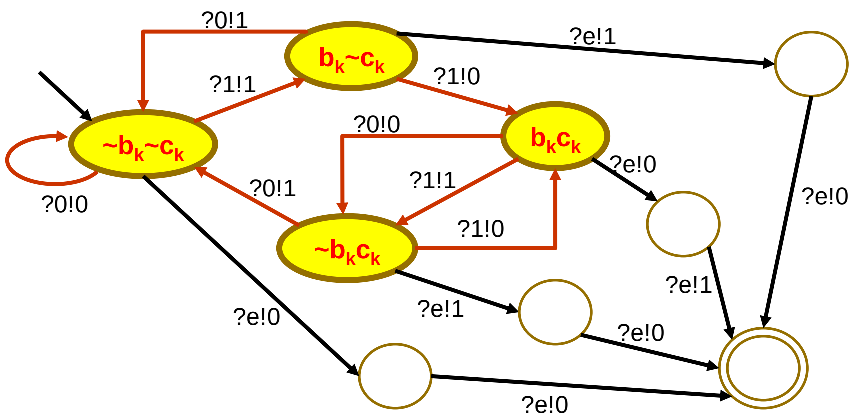


State-transition graph for automata - an exercise run for ee1011(11)



$a_{n+2}a_{n+1}\ldots a_1a_0 = 3*(eeb_nb_{n-1}\ldots b_1b_0)$
➔ $a_{k+1} = b_{k+1} + b_k + c_k$

carry



State-transition graph for automata

- How to construct an automata for $c \in N$,

  $c*(b_n b_{n-1} \ldots b_1 b_0)$

  Need $(\lceil \log 2(c) \rceil) * 2^{\lceil \log 2(c) \rceil} + 1$ states!
- How to construct an automata for

  $a_n a_{n-1} \ldots a_1 a_0 + b_n b_{n-1} \ldots b_1 b_0$?

  Can you do this ?
- How to construct an automata for

  $\sum c_k * (b_{n,k} b_{n-1,k} \ldots b_{1,k} b_{0,k})$?

106

Kripke Structures - composition for a concurrent system

Given $A_i = \langle S_i, S_{i,0}, R_i, L_i \rangle$, $1 \leq i \leq n$

Cartesian Product of $A_1, A_2, \ldots, A_n$,

$A = \langle S, S_0, R, L \rangle$

$S : S_1 \times S_2 \times \ldots \times S_n$

$S_0 : S_{1,0} \times S_{2,0} \times \ldots \times S_{n,0}$

$R([s_1, \ldots, s_{j-1}, s_j, s_{j+1}, \ldots, s_n], [s_1, \ldots, s_{j-1}, s'_j, s_{j+1}, \ldots, s_n])$
- $(s_i, s_i') \in R_i$
- *According to the interleaving semantics, one process transition at a moment*

$L([s_1, s_2, \ldots, s_n]) = L_1(s_1) \cup L_2(s_2) \cup \ldots \cup L_n(s_n)$

107

Kripke Structures
- Cartesian product method

1. Construct all the vectors of component process states
2. Eliminate all those inconsistent vectors according to invariance condition
3. Draw arcs from vectors to vectors according to process transitons

- *Very often creates many unreachable states*

Kripke structure
- Practical algorithm for construction

Given $A=\langle S, S_0, R, L\rangle$

- Usually only S_0, R, L are given.
- We may want to construct S.
- Usually S is too big to construct.

Kripke Structures
- on-the-fly method

1. Starting from the initial states (or goal states in backward analysis)
2. Step by step, add states that is reachable from those already reached, until no more new reachable states are generated.

- *Tedious but may result in much smaller reachable state-space reprsentation.*

Kripke Structures
- forward reachability analysis

- Use strongest postcondition to compute state-spaces forward reachable from initial states
- Can only be used for safety analysis
- Very often can lead to larger state-space represenation
- Very often can lead to unnecessary total ordering enumeration
  - Need symmetry reduction and partial-order reduction

Kripke Structures
- backward reachability analysis

- Use weakest precondition to compute state-spaces backward reachable from goal states
- The mandatory method for model-checking
- More like refutation
- Very often can lead to smaller state-space represenation
- Very often can lead to less total ordering enumeration

Kripke Structure
- propositions

Given by the valuation of the variables defining the states. Possible propositions

$pc_0 = l_0, \ldots, pc_0 = l_3$

$pc_1 = m_0, \ldots, pc_1 = m_3$

$turn = 0, turn = 1$

Clearly the proposition $pc_0 = l_0$ is true in any state of the form $\langle pc_0 = l_0, pc_1 = ?, turn = ??\rangle$

This clarifies the labeling function *L* in Kripke Structure

Kripke Structure
- system properties

- Propositions can be combined to state interesting properties

  It is never the case that $pc_0 = l_2$ and $pc_1 = m_2$

  The above is the mutual exclusion property. We will study a logic for describing properties in next class.

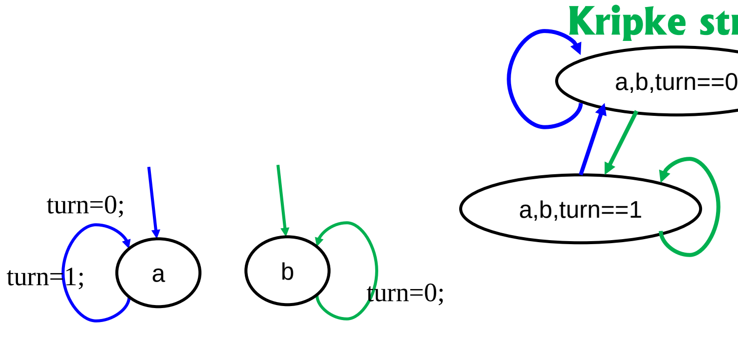
Kripke Structure
- fairness in a concurrent system

In a concurrent system, there could be several independent modules with independent descriptions.

- How can we construct the Kripke structure for global behavior description ?
- How can we run the modules *fairly* ?
  - Is there a module that never gets execution in interleaving semantics ?
  - Is an unfair execution meaningless ?

Fairness in concurrent systems

Semantics as Kripke structure



state-transition graphs

Kripke Structure
- fairness in a concurrent system

- Proc0 manipulates X
- Proc1 manipulates Y
- In the global state <X=0, Y=1>
  - Proc0 or Proc1 could make a move.
  - We allow the behavior that Proc1 always makes a move (self-loop)
  - System is stuck at <X= 0, Y=1>
  - Unfair execution !

Fair Kripke Structures

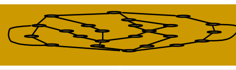
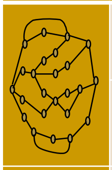
- $M = (S, S_0, R, L, F)$
  - S, S_0, R, L as before.
  - $F \subseteq 2^S$ is a set of fairness constraints.
  - Each element of F is a set of states which must occur infinitely often in any execution path.
- In our example, F = {{<X=1,Y=1>}}
  - Avoid getting stuck at <X=0,Y=1> or <X=1, Y = 0>

Kripke structure - verification

- safety analysis
  - Can the system be always safe ?
  - Can a risk state happen ?
- liveness analysis
  - Can the job be done sometimes ?
  - Can the job be prevented from been done ?
- bisimulation checking
  - Are two Kripke structures the same transition by transition ?
- simulation checking
  - Can one Kripke structure match every transition by the another ?
- language inclusion
  - Are all traces of one Kripke structure also ones of another ?

Model-checking
- frameworks in our lecture

F: set of fairness assumptions.
✓: known;
☑: discussed in the lecture

Spec / Model							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≠∅							✓	✓

2009/11/25 stopped here.

Kripke structure
- safety analysis

Given

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

can η be false at some state along some runs ?

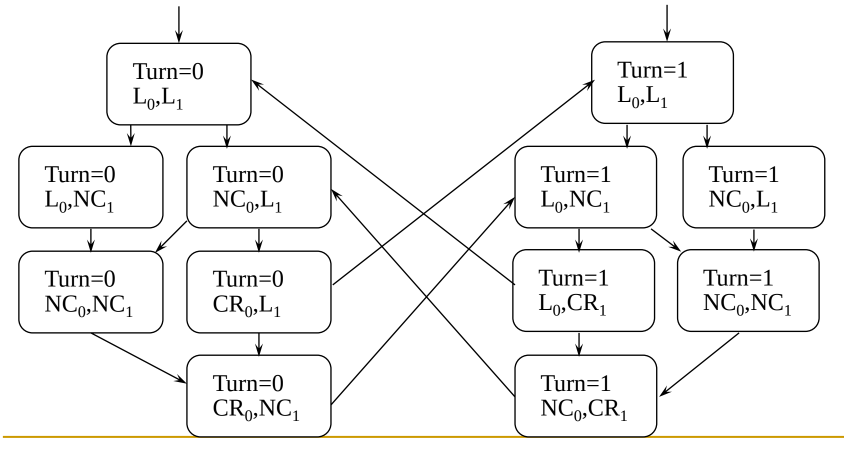
Example:

Can the engine stall ?

Can the boiler be overheated ?

Kripke structure
- safety analysis

☐ ¬(PC0=CR0∧PC1=CR1) is an invariant!



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.'
}
```

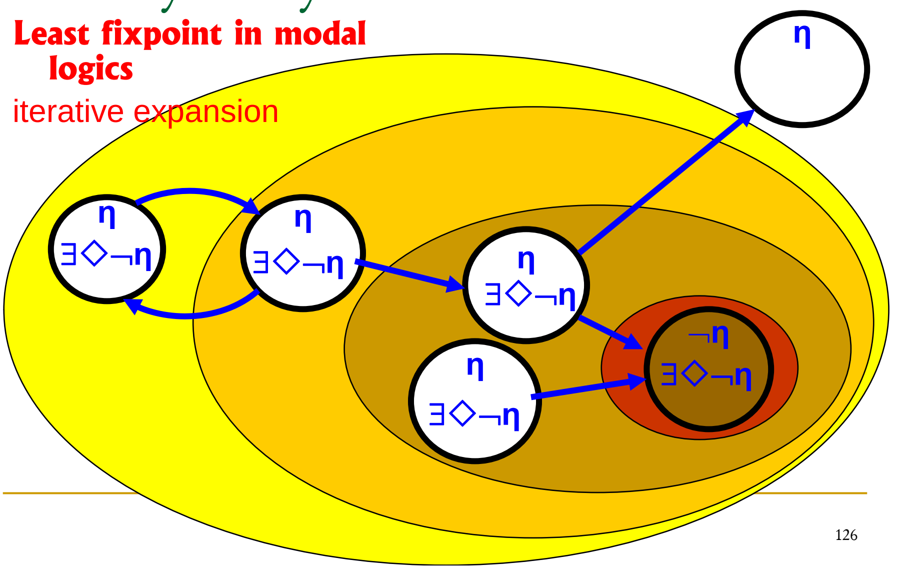
A notation for the possibility of $\neg\eta$

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

Kripke structure
- safety analysis

Least fixpoint in modal logics

iterative expansion



Kripke structure
- *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[]* ?

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

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.
- $\neg$*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 and j is not in the 2^{nd} round.
- *2nd[i]* iff suspect i to be in 2nd round investigation.

What is the maximal solution to $\neg$ *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 $\neg$ *2nd[]* ?

$\neg$ *2nd[i]* ➔ $\neg$*Liar[i]* $\wedge$ $\exists$*j*$\neq$*i*($\neg$*2nd[j]* $\wedge$ *Loyal-to[i,j]*)

In comparison:

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

~~$\neg$ *2nd[i]* $\equiv$ $\neg$*Liar[i]* $\wedge$ $\forall$*j*$\neq$*i*($\neg$*2nd[j]* ➔ *Loyal-to[i,j]*)~~

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

CTL
- symbolic model-checking with BDD

- In a Kripke structure, 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.

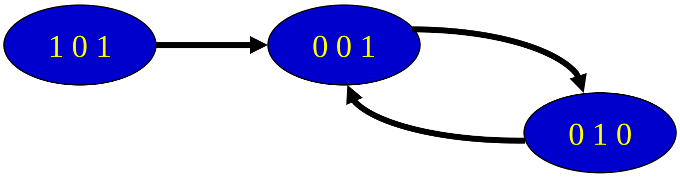
  a Boolean function with *n* binary variables

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

CTL - symbolic model-checking with Propositioal logics

Example:

$x_1\ \ x_2\ \ x_3$

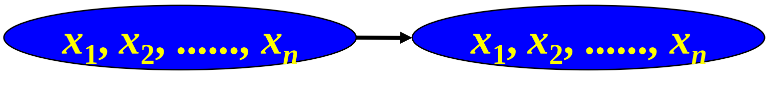


$S(x_1, x_2, x_3) = (x_1 \wedge \neg x_2 \wedge x_3)$
$\vee\ (\neg x_1 \wedge \neg x_2 \wedge x_3)$
$\vee\ (\neg x_1 \wedge x_2 \wedge \neg x_3)$

CTL - symbolic model-checking with Propositioal logics

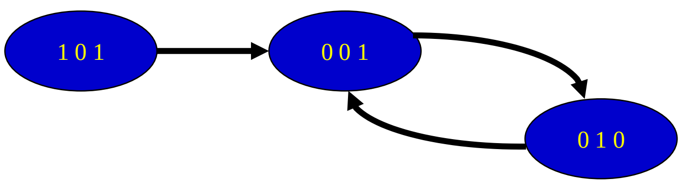
State transition relation as a logic funciton with <u>2*n* parameters</u>

$R(x_1, x_2, \ldots\ldots, x_n, x'_1, x'_2, \ldots\ldots, x'_n)$



CTL - symbolic model-checking with Propositioal logics

$x_1 \ x_2 \ x_3 \ x'_1 \ x'_2 \ x'_3$



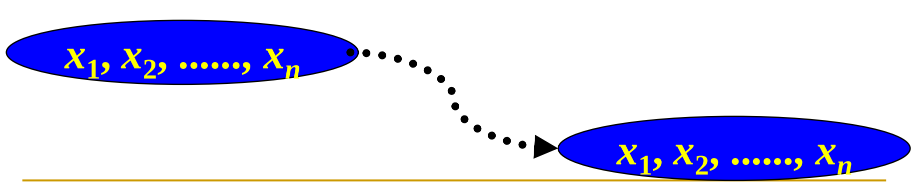
$$R(x_1, x_2, x_3, x'_1, x'_2, x'_3) =$$
$$(x_1 \wedge \neg x_2 \wedge x_3 \wedge \neg x'_1 \wedge \neg x'_2 \wedge x'_3)$$
$$\vee \quad (\neg x_1 \wedge \neg x_2 \wedge x_3 \wedge \neg x'_1 \wedge x'_2 \wedge \neg x'_3)$$
$$\vee \quad (\neg x_1 \wedge x_2 \wedge \neg x_3 \wedge \neg x'_1 \wedge \neg x'_2 \wedge x'_3)$$

CTL - symbolic model-checking with Propositioal logics

Path relation also as a logic funciton

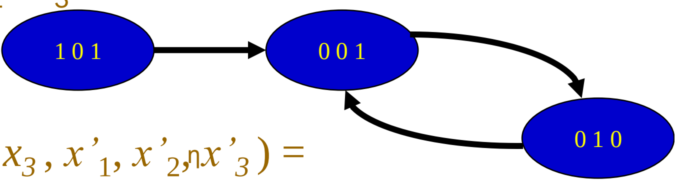
with <u>2*n* parameters</u>

reach($x_1, x_2, \ldots\ldots, x_n, x'_1, x'_2, \ldots\ldots, x'_n$)



CTL - symbolic model-checking with Propositioal logics

$x_1\ x_2\ x_3\ x'_1\ x'_2\ x'_3$



$$\mathrm{reach}(x_1, x_2, x_3, x'_1, x'_2, x'_3) =$$

$$\begin{aligned}
& (x_1\wedge\neg x_2\wedge x_3\wedge\neg x'_1\wedge\neg x'_2\wedge x'_3)\\
\vee\ & (x_1\wedge\neg x_2\wedge x_3\wedge\neg x'_1\wedge x'_2\wedge\neg x'_3)\\
\vee\ & (\neg x_1\wedge\neg x_2\wedge x_3\wedge\neg x'_1\wedge x'_2\wedge\neg x'_3)\\
\vee\ & (\neg x_1\wedge x_2\wedge\neg x_3\wedge\neg x'_1\wedge\neg x'_2\wedge x'_3)\\
\vee\ & (\neg x_1\wedge\neg x_2\wedge x_3\wedge\neg x'_1\wedge\neg x'_2\wedge x'_3)\\
\vee\ & (\neg x_1\wedge x_2\wedge\neg x_3\wedge\neg x'_1\wedge x'_2\wedge\neg x'_3)
\end{aligned}$$

Symbolic safety analysis

- I : initial condition with parameters

  $x, x_2, \ldots\ldots, x_n$

- η : safe condition with parameters

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

If $I\wedge\neg(\eta\uparrow)\wedge\ \mathrm{reach}(x_1, x_2, \ldots\ldots, x_n, x'_1, x'_2, \ldots\ldots, x'_n)$

is not false,

- a risk state is reachable.
- *the system is not safe.*

change all umprimed variables in η to primed.

Symbolic safety analysis
- construction of reach(x_1,......, x_n, x'_1,......, x'_n)

$$
\begin{aligned}
& R(x_1,\ldots\ldots, x_n, x'_1,\ldots\ldots, x'_n) \\
& \vee \exists y_1,\ldots\ldots, \exists y_n \,(\; R(x_1,\ldots\ldots, x_n, y_1,\ldots\ldots, y_n) \\
& \qquad\qquad\qquad\quad \wedge \mathrm{reach}(y_1,\ldots\ldots, y_n, x'_1,\ldots\ldots, x'_n) \\
& \qquad\qquad\qquad\quad)
\end{aligned}
$$

➔ reach(x_1,......, x_n, x'_1,......, x'_n)

This is a least fixpoint for backward analysis.

Symbolic safety analysis
- construction of reach(x_1,......, x_n, x'_1,......, x'_n)

$$
\begin{aligned}
& R(x_1,\ldots\ldots, x_n, x'_1,\ldots\ldots, x'_n) \\
& \vee \exists y_1,\ldots\ldots, \exists y_n \,(\; \mathrm{reach}(x_1,\ldots\ldots, x_n, y_1,\ldots\ldots, y_n) \\
& \qquad\qquad\qquad\quad \wedge \mathrm{reach}(y_1,\ldots\ldots, y_n, x'_1,\ldots\ldots, x'_n) \\
& \qquad\qquad\qquad\quad)
\end{aligned}
$$

➔ reach(x_1,......, x_n, x'_1,......, x'_n)

This is *another* least fixpoint for speed-up.

Symbolic safety analysis
- construction of reach(x_1,......, x_n, x'_1,......, x'_n)

R(x_1,......, x_n, x'_1,......, x'_n)

$\vee\exists y_1$,......, $\exists y_n$ (reach(x_1,......, x_n, y_1,......, y_n)

$\wedge$ R(y_1,......, y_n, x'_1,......, x'_n)

)

➔ reach(x_1,......, x_n, x'_1,......, x'_n)

This is *another* least fixpoint for forward analysis.

140

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 $\neg\eta(x_0,x_1,\ldots,x_n)$
- the initial state set as $I(x_0,x_1,\ldots,x_n)$
- the transition set as $R(x_0,x_1,\ldots,x_n,x'_0,x'_1,\ldots,x'_n)$

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

$b_0 = \neg\eta(x_0,x_1,\ldots,x_n) \wedge S(x_0,x_1,\ldots,x_n)$; $k = 1$;

repeat

$b_k = b_{k-1} \vee \exists x'_0 \exists x'_1 \ldots \exists x'_n (R(x_0,x_1,\ldots,x_n,x'_0,x'_1,\ldots,x'_n) \wedge (b_{k-1}\uparrow))$;

k = k +1;

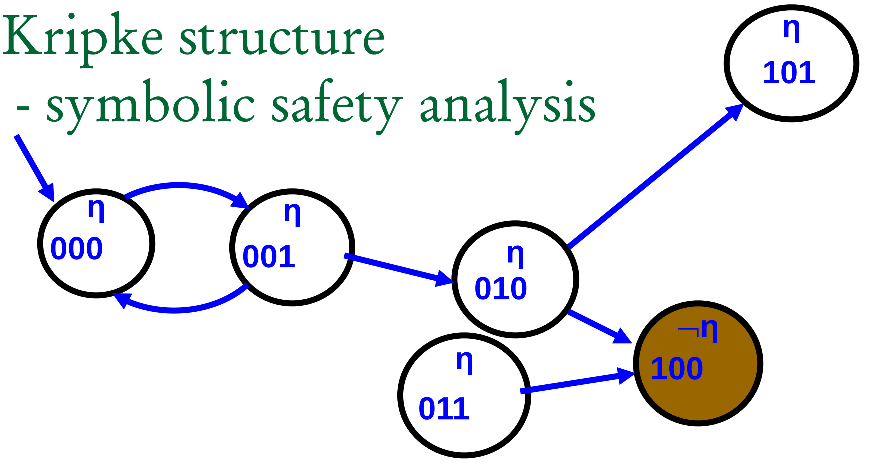
until $b_k \equiv b_{k-1}$;

a least fixpoint procedure

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

141

Kripke structure
- symbolic safety analysis

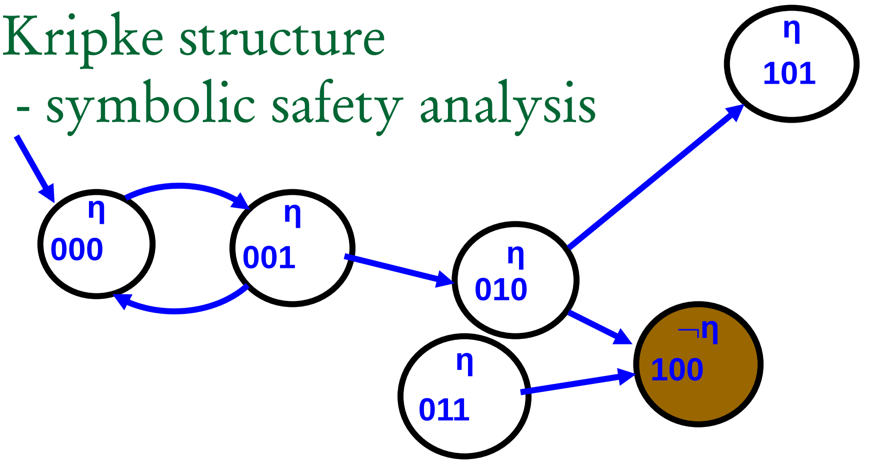


states: $S(x,y,z) \equiv (\neg x \wedge \neg y \wedge \neg z) \vee (\neg x \wedge \neg y \wedge z) \vee (\neg x \wedge y \wedge \neg z)$
$\vee (\neg x \wedge y \wedge z) \vee (x \wedge \neg y \wedge \neg z) \vee (x \wedge \neg y \wedge z)$
$\equiv (\neg x) \vee (x \wedge \neg y)$

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

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

Kripke structure
- symbolic safety analysis



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

2009/12/02 stopped here.

144

Symbolic safety analysis (backward)

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

$b_1 = b_0 \vee \exists x'\exists y'\exists z'(R(x,y,z,x',y',z') \wedge b_0\uparrow)$

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

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

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

$b_2 = b_1 \vee \exists x'\exists y'\exists z'(R(x,y,z,x',y',z') \wedge b_1\uparrow)$

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

$b_3 = b_2 \vee \exists x'\exists y'\exists z'(R(x,y,z,x',y',z') \wedge b_2\uparrow)$

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

$b_4 = b_3 \vee \exists x'\exists y'\exists z'(R(x,y,z,x',y',z') \wedge b_3\uparrow)$

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

fixpoint

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

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

145

老子道德經四十一章

大音希聲

大象無形

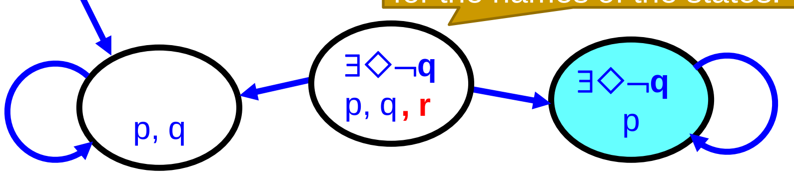
道隱無名

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 $\neg\eta(x_0, x_1, \ldots, x_n)$
- the initial state set as $I(x_0, x_1, \ldots, x_n)$
- the transition set as $R(x_0, x_1, \ldots, x_n, x'_0, x'_1, \ldots, x'_n)$

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

$f_k = f_{k-1} \vee (\exists x_0 \exists x_1 \ldots \exists x_n (R(x_0, x_1, \ldots, x_n, x'_0, x'_1, \ldots, x'_n) \wedge f_{k-1}))\downarrow$;

$k = k + 1$;

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

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

Symbolic safety analysis (forward)

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

$f_1 = f_0 \vee (\exists x \exists y \exists z (R(x,y,z,x',y',z') \wedge f_0))\downarrow$

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

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

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

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

$f_2 = f_1 \vee (\exists x \exists y \exists z (R(x,y,z,x',y',z') \wedge f_1))\downarrow = (\neg x \wedge \neg y) \vee (\neg x \wedge y \wedge \neg z)$

$f_3 = f_2 \vee (\exists x \exists y \exists z (R(x,y,z,x',y',z') \wedge f_2))\downarrow = (\neg y) \vee (\neg x \wedge y \wedge \neg z)$

$f_4 = f_3 \vee (\exists x \exists y \exists z (R(x,y,z,x',y',z') \wedge f_3))\downarrow = (\neg y) \vee (\neg x \wedge y \wedge \neg z)$

fixpoint

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

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

Bounded model-checking

The value of x_n at state k.

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

- the state set as a proposition formula: $S(x_{0,k},x_{1,k},\ldots,x_{n,k})$
- the risk state set as $\neg\eta(x_{0,k},x_{1,k},\ldots,x_{n,k})$
- the initial state set as $I(x_{0,0},x_{1,0},\ldots,x_{n,0})$
- the transition set as $R(x_{0,k-1},x_{1,k-1},\ldots,x_{n,k-1},x_{0,k},x_{1,k},\ldots,x_{n,k})$

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

repeat

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

$k = k+1$;

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

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

Bounded model-checking

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

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

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

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

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

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

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

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

$)$

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

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

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

Transition relation
- from state-transition graphs

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

r_k: when (τ_k) may $y_{k,0}=d_0$; $y_{k,1}=d_1$; …; $y_{k,nk}=d_{nk}$;

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

152

Transition relation from GCM rules.

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

r_1: when (x<y&& y>2) may y=x+y; x=3;

r_2: when (z>=2) may y=x+1; z=0;

r_3: when (x<2) may x=0;

$$R(x_0,x_1,\ldots,x_n,x'_0,x'_1,\ldots,x'_n)$$
$$\equiv (x<y \wedge y>2 \wedge y'==x+y \wedge x'==3 \wedge z'==z)$$
$$\vee(z>=2 \wedge y'==x+1 \wedge z'==0 \wedge x'==x)$$
$$\vee(x<2 \wedge x'==0 \wedge y'==y \wedge z'==z)$$

153

Transition relation - from state-transition graphs

In general, transition relation is expensive to construct.

Can we do the following state-space construction

$\exists x'_0 \exists x'_1 \ldots \exists x'_n (R(x_0,x_1,\ldots,x_n,x'_0,x'_1,\ldots,x'_n) \wedge (b_{k-1}\uparrow))$

directly with the GCM rules ?

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

On-the-fly precondition calculation with GCM rules.

$\exists x'_0 \exists x'_1 \ldots \exists x'_n (R(x_0,x_1,\ldots,x_n,x'_0,x'_1,\ldots,x'_n) \wedge (b\uparrow))$

$\equiv \vee_{k\in[1,m]} (\tau_k \wedge \exists y_{k,0} \exists y_{k,1} \ldots \exists y_{k,nk} (b \wedge \wedge_{h\in[0,nk]} y_{k,h}{=}{=}d_h))$

```
pre(b) {
  r = false;
  for k = 1 to m, {
    let f = b;
    for h=nk to 0, f = ∃y_{k,h}(f ∧ y_{k,h}==d_h);
    r = r ∨ (τ_k ∧ f);
  }
  return (r);
}
```

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: when (τ_k) may $y_{k,0}=d_0$; $y_{k,1}=d_1$; …; $y_{k,nk}=d_{nk}$;

$$\exists x'_0 \exists x'_1 \ldots \exists x'_n (R(x_0,x_1,\ldots,x_n,x'_0,x'_1,\ldots,x'_n) \wedge (b\uparrow))$$
$$\equiv \bigvee_{k\in[1,m]} \Big(\tau_k \wedge \exists y_{k,0} \exists y_{k,1} \ldots \exists y_{k,nk} \big(b \wedge \bigwedge_{h\in[0,nk]} y_{k,h}==d_h \big) \Big)$$

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: when (x<y&& y>2) may y=z; x=3;

r_2: when (z>=2) may y=x+1; z=7;

r_3: when (x<2) may z=0;

B

$$\exists x'_0 \exists x'_1 \ldots \exists x'_n (R(x_0,x_1,\ldots,x_n,x'_0,x'_1,\ldots,x'_n) \wedge (x<4 \wedge z>5)\uparrow)$$
$$\equiv (x<y \wedge y>2 \wedge \exists y \exists x(x<4 \wedge z>5 \wedge y==z \wedge x==3))$$
$$\vee (z>=2 \wedge \exists y \exists z(x<4 \wedge z>5 \wedge y==x+1 \wedge z==7))$$
$$\vee (x<2 \wedge \exists z(x<4 \wedge z>5 \wedge z==0))$$
$$\equiv (x<y \wedge y>2 \wedge z>5) \vee (z>=2 \wedge x<4) \vee (x<2 \wedge \exists z(\text{false}))$$
$$\equiv (x<y \wedge y>2 \wedge z>5) \vee (z>=2 \wedge x<4)$$

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: when (τ_k) may 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

158

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: when (τ_k) may s_k;

What is pre(s,b) ?

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

- pre($x = E$;, b) $\equiv$ 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/5] ≡ 5==y+2 ∧ 5<4∧z>5

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

159

On-the-fly precondition calculation with GCM rules.

Given a set of rules r_1, r_2, …, r_m of the form

r_k: when (τ_k) may s_k;

What is pre(s,b) ?

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

- pre($x = E$;, b) $\equiv$ b[x/E]
- pre($s_1 s_2$, b) $\equiv$ pre(s_1, pre(s_2, b))
- pre(if (B) $s_1$else s_2) $\equiv$ (B$\wedge$pre(s_1, b))$\vee$($\neg$B$\wedge$pre(s_2,b))
- pre(while (B) s, b) $\equiv$ ….

Ex. the precondition to x=x+z;
(x==y+2 $\wedge$ x<4$\wedge$z>5) [x/x+z]
$\equiv$ x+z==y+2 $\wedge$ x+z<4$\wedge$z>5

On-the-fly precondition calculation with GCM rules.

Given a set of rules r_1, r_2, …, r_m of the form

r_k: when (τ_k) may s_k;

What is pre(s,b) ?

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

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

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

On-the-fly precondition calculation with GCM rules.

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

$w_0 = \neg K \wedge b$; $k = 1$;

also a least fixpoint procedure

repeat

  $w_k = w_{k-1} \vee (K \wedge pre(s, w_{k-1}))$;

  $k = k + 1$;

until $w_k \equiv w_{k-1}$;

return w_k;

Precondition to b through while (K) s;

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

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

while (x < y) x = 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 (K) s, b)

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

$w_0 = K; k = 1;$

repeat

a *greatest* fixpoint procedure

$w_k = K \wedge pre(s, w_{k-1});$

$k = k + 1;$

until $w_k \equiv w_{k-1};$

return w_k;

164

Precondition to b through while (K) s;

$w_0 = K; k = 1;$
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!

165

Precondition to b through while (K) s;

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

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 (K) s;

```
w0 = ¬K∧b; k = 1;
repeat
   wk = wk-1∨(K∧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>0) x = x+1;

L_1 computation.

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

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

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

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

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

Kripke structure - liveness analysis

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

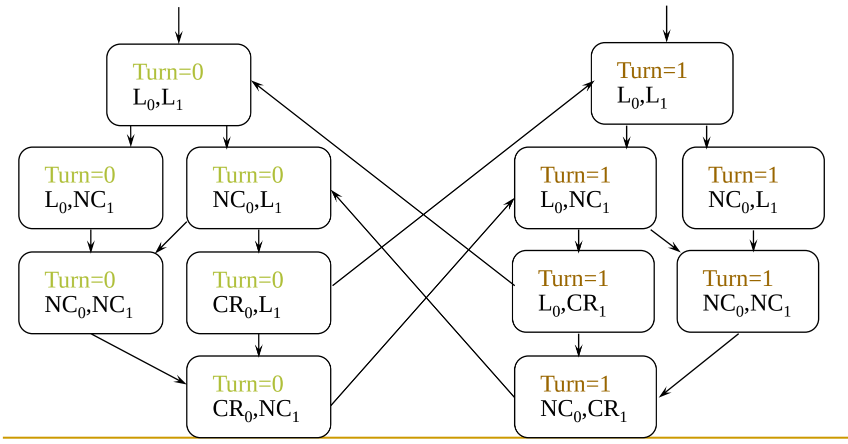
- all states in the cycle are $\neg\eta$
- there is a $\neg\eta$ path from a state in S_0 to the cycle.

Solutions in graph theory

- strongly connected components (SCC)

Kripke structure
- liveness analysis

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



Kripke structure
- liveness analysis

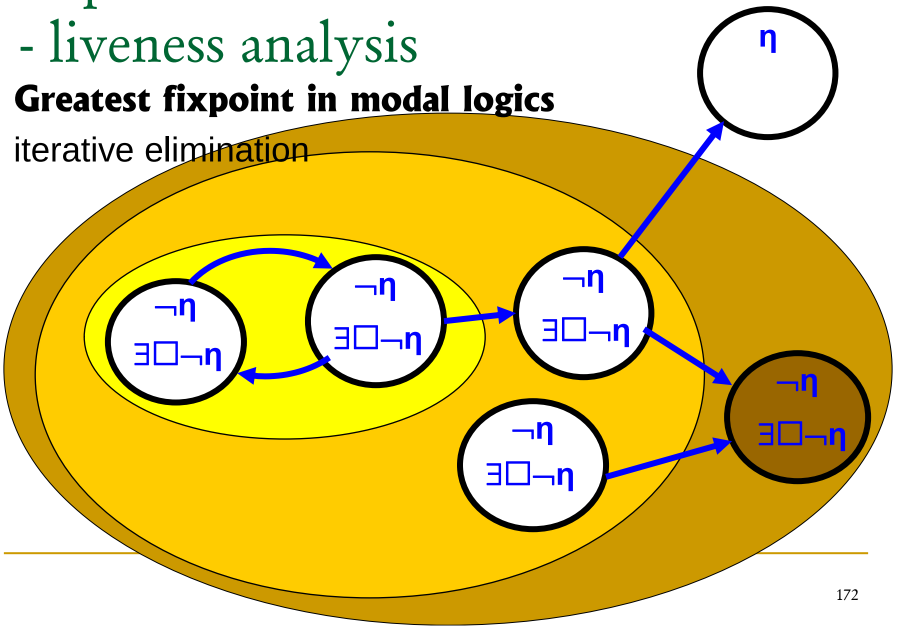
liveness(**η**) /* using greatest fixpoint algorithm */ {
  for all s, if $\neg\eta \in L(s)$, $L(s)=L(s)\cup\{\exists\Box\neg\eta\}$;
  repeat {
    for all s, if $\exists\Box\neg\eta \in L(s)$ and $\forall(s,s')(\exists\Box\neg\eta \notin L(s))$,
      $L(s)=L(s) - \{\exists\Box\neg\eta\}$;
  } until no more changes to L(s) for any s.
  if there is an $s_0 \in S_0$ with $\exists\Box\neg\eta \in L(s_0)$,
    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



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 $\neg\boldsymbol{\eta}(x_0,x_1,\ldots,x_n)$
- the initial state set as $I(x_0,x_1,\ldots,x_n)$
- the transition set as $R(x_0,x_1,\ldots,x_n,x'_0,x'_1,\ldots,x'_n)$

$b_0 = \neg\boldsymbol{\eta}(x_0,x_1,\ldots,x_n) \wedge S(x_0,x_1,\ldots,x_n)$; $k = 1$;

repeat

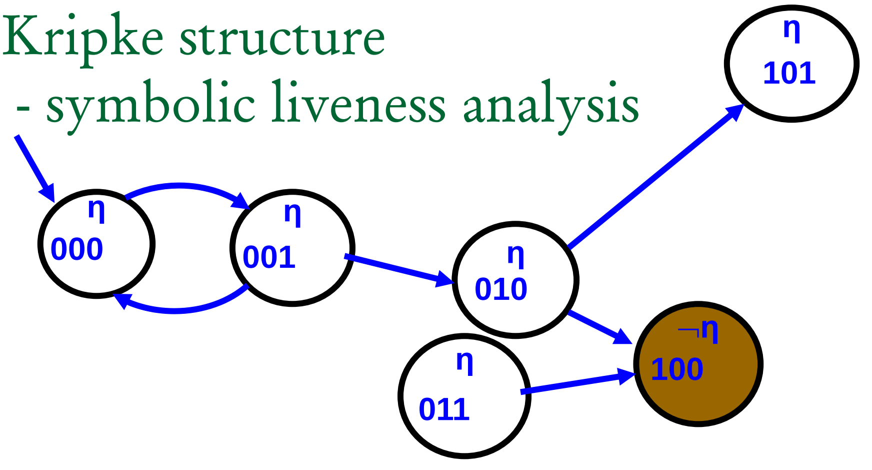
$b_k = b_{k-1}\wedge\exists x'_0\exists x'_1\ldots\exists x'_n(R(x_0,x_1,\ldots,x_n,x'_0,x'_1,\ldots,x'_n)\wedge b'_{k-1})$;

$k = k+1$;

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';

Kripke structure
- symbolic liveness analysis

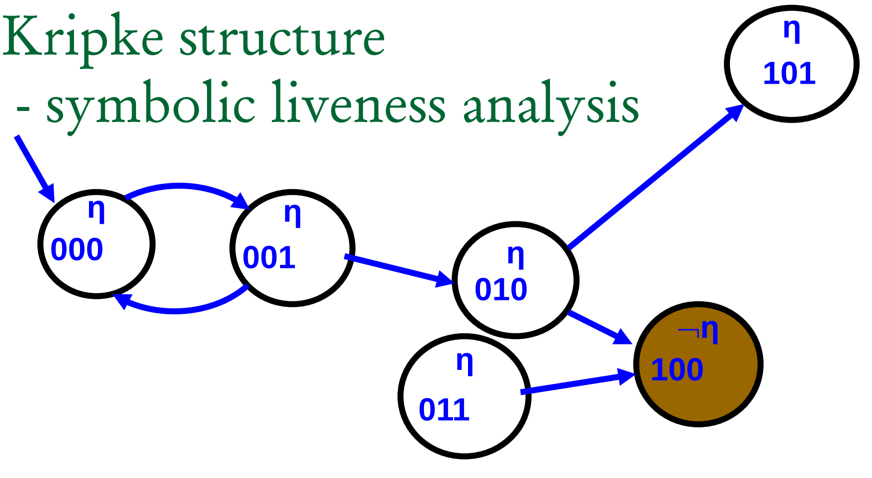


states: $S(x,y,z) \equiv (\neg x \wedge \neg y \wedge \neg z) \vee (\neg x \wedge \neg y \wedge z) \vee (\neg x \wedge y \wedge \neg z)$
$\vee (\neg x \wedge y \wedge z) \vee (x \wedge \neg y \wedge \neg z) \vee (x \wedge \neg y \wedge z)$
$\equiv (\neg x) \vee (x \wedge \neg y)$

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

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

Kripke structure
- symbolic liveness analysis



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

Symbolic liveness analysis

b0 = $\neg\boldsymbol{\eta}(x,y,z) \equiv (\neg x)\vee(x\wedge\neg y\wedge z)$

b1 = b0 $\wedge \exists x'\exists y'\exists z'(R(x,y,z,x',y',z')\wedge b0')$

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

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

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

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

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

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

b2 = b1 $\wedge \exists x'\exists y'\exists z'(R(x,y,z,x',y',z')\wedge b1')$

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

b3 = b2 $\wedge \exists x'\exists y'\exists z'(R(x,y,z,x',y',z')\wedge b2')$

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

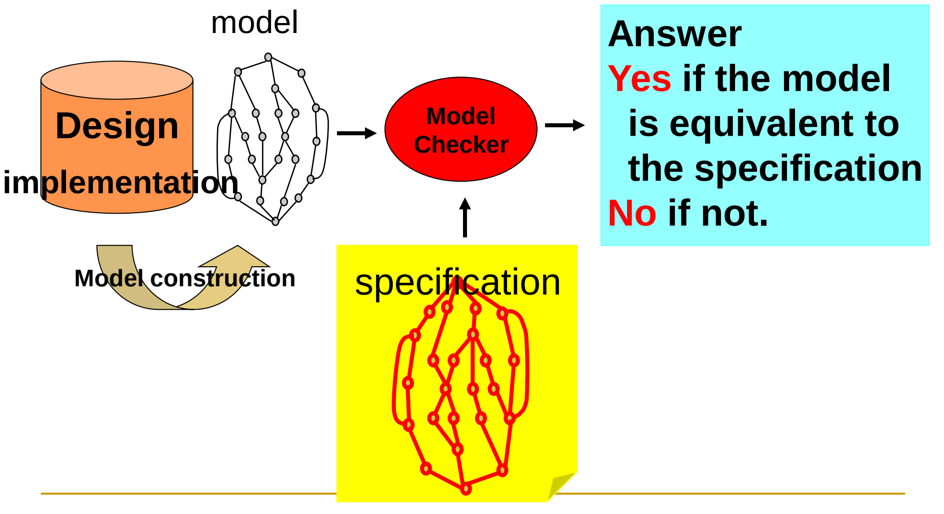
$= \neg x\wedge\neg y$

fixpoint

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

2009/12/16 stopped here.

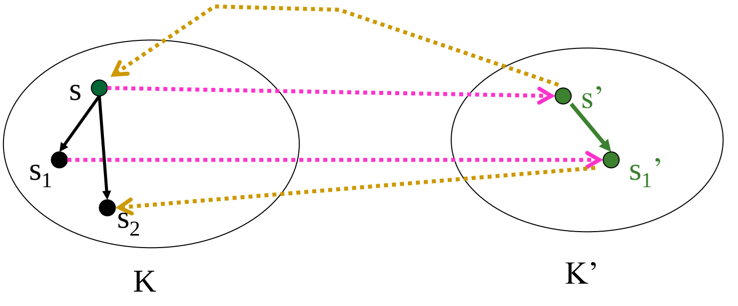
Bisimulation Framework



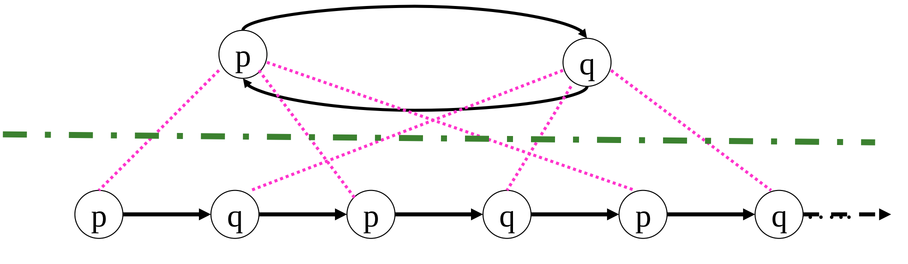
Bisimulation-checking

- $K = (S, S_0, R, L)$
  $K' = (S', S_0', R', L')$
- Note K and K' use the same set of atomic propositions P.
- $B \in S \times S'$ 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)

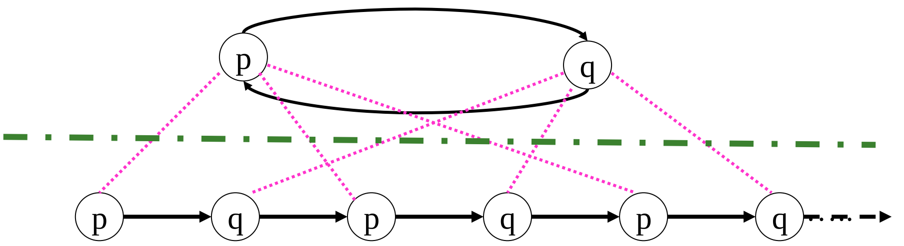
Bisimulations



Examples

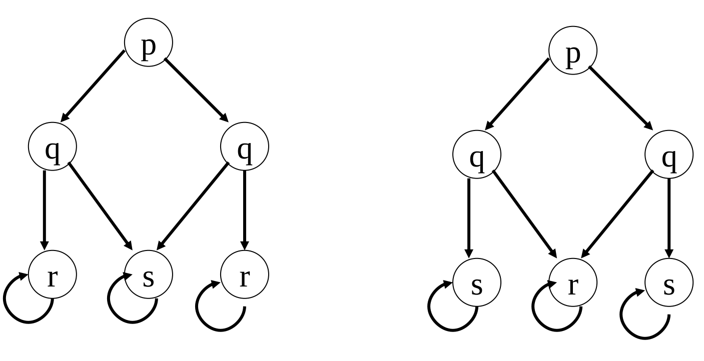


Examples

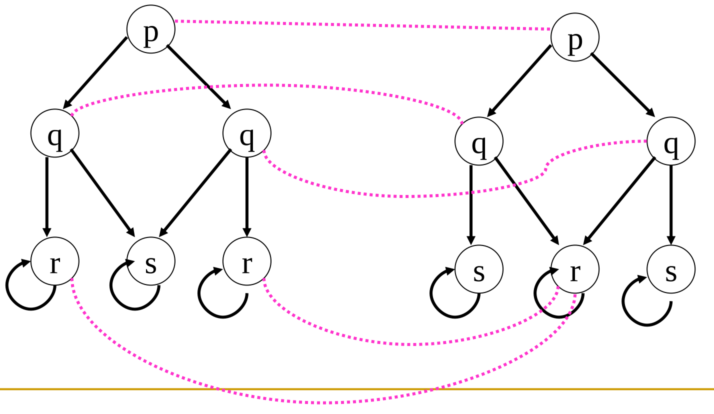


Unwinding preserves bisimulation

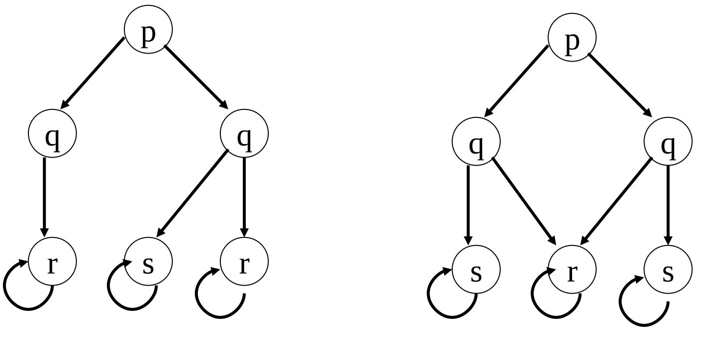
Examples



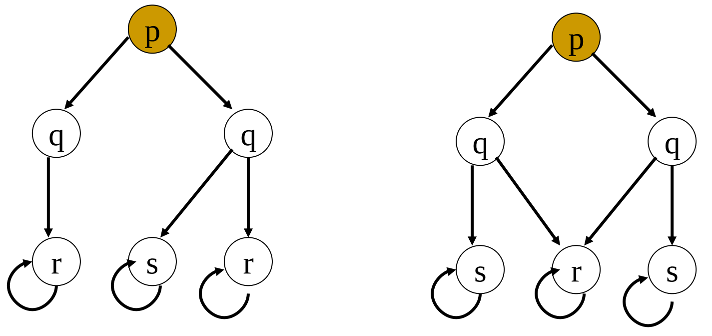
Examples



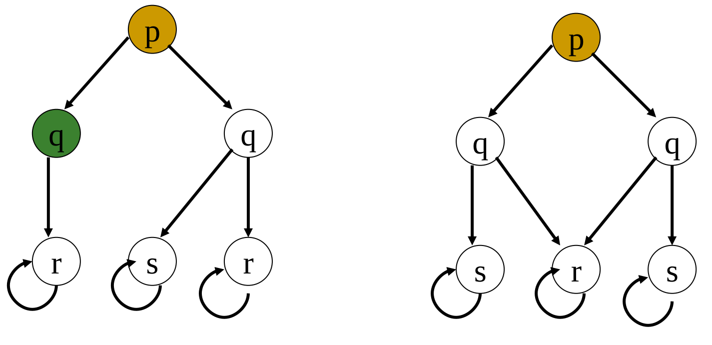
Examples



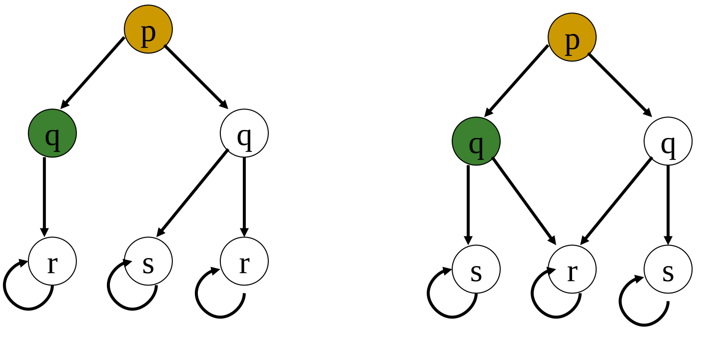
Examples



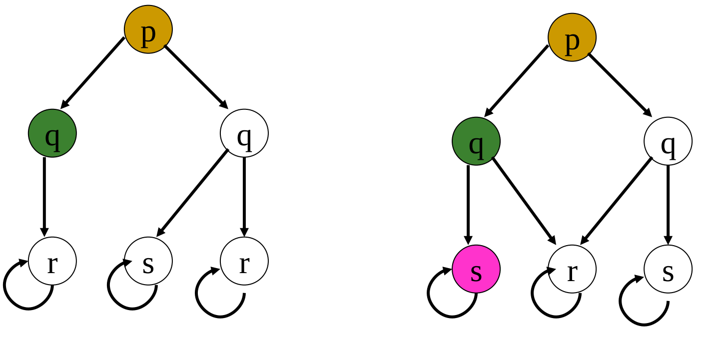
Examples



Examples



Examples



Bisimulations

- $K = (S, S_0, R, L)$
- $K' = (S', S_0', R', L')$
- K and K' are bisimilar (bisimulation equivalent) iff there exists a bisimulation relation $\mathcal{B} \subseteq S \times S'$ between K and K' such that:
  - For each s_0 in S_0 there exists s_0' in S_0' such that $\mathcal{B}(s_0 , s_0')$.
  - For each s_0' in S_0' there exists s_0 in S_0 such that $\mathcal{B}(s_0 , s_0')$.

The Preservation Property.

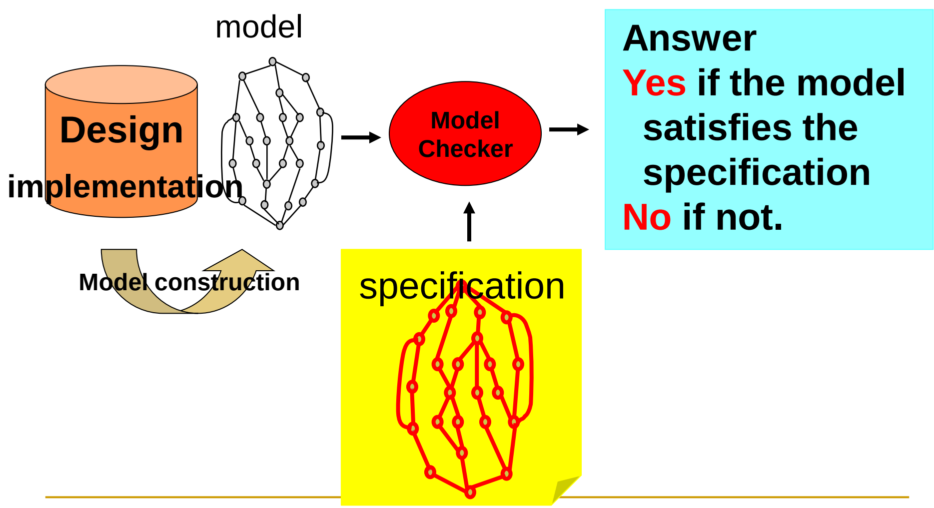
- $K = (S, S_0, R, AP, L)$
  $K' = (S', S_0', R', AP, L')$
- $\mathcal{B} \subseteq S \times S'$, a bisimulation.
- Suppose $\mathcal{B}(s, s')$.

FACT: For any CTL* formula ψ (over AP),

$$K, s \models \psi \text{ iff } K', s' \models \psi.$$

If K' is smaller than K this is worth something.
→ abstraction for space reduction

Simulation Framework



192

Simulation-checking

- K = (S, S_0, R, L)
  K'= (S', S_0', R', L')
- Note K and K' use the same set of atomic propositions AP.
- $B \subseteq S \times S'$ 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)

193

Simulations

- $K = (S, S_0, R, L)$
- $K' = (S', S_0', R', L')$
- 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 $B(s_0, s_0')$.

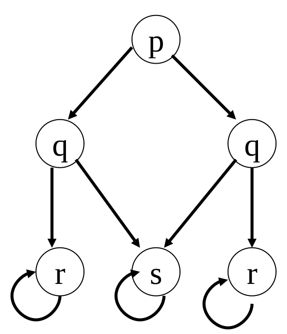
Bisimulation Quotients

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

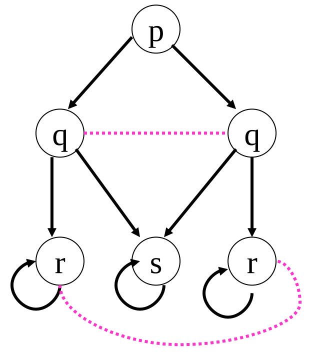
Bisimulation Quotient

- $K = (S, S_0, R, L)$
- $[s] = \{s' \mid s \, B \, s'\}$.
- $K' = K / B = (S', S'_0, R', L')$.
  - $S' = \{[s] \mid s \in S\}$
  - $S'_0 = \{[s_0] \mid s_0 \in S_0\}$
  - $R' = \{([s], [s']) \mid R(s_1, s_1') , s_1 \in [s] , s_1' \in [s']\}$
  - $L'([s]) = L(s)$.

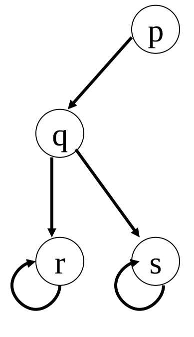
Examples



Examples



Examples



Abstractions

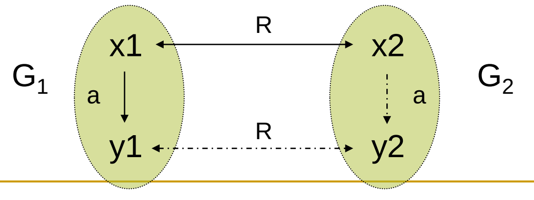
- Bisimulations don't produce often large reduction.
- Try notions such as simulations, data abstractions, symmetry reductions, partial order reductions etc.
- Not all properties may be preserved.
- They may not be preserved in a strong sense.

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

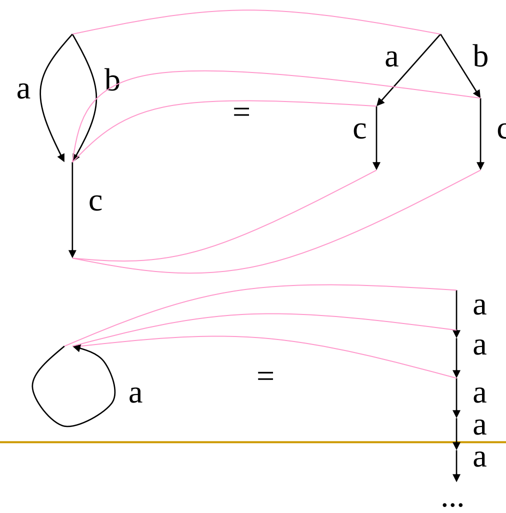
A *bisimulation* is a relation R between nodes s.t. both R and R^{-1} are simulations

Set Semantics for Semistructured Data

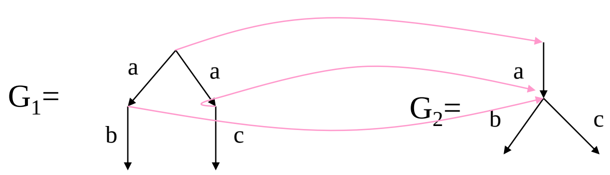
Definition Two rooted graphs G_1, G_2 are equal if there exists a bisimulation R from G_1 to G_2 such that $(\text{root}(G_1), \text{root}(G_2)) \in R$

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

Examples of Bisimilar Graphs



Examples of non-Bisimilar Graphs



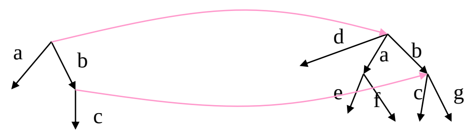
- This is a *simulation* but not a *bisimulation*
  - Why ?
- Notice: G_1, G_2 have the same sets of *paths*

Examples of Simulation

- Simulation acts like "subset"

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

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

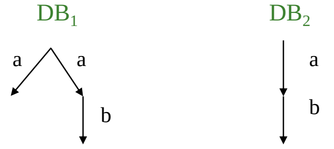


- Question:
- if $DB_1 \subseteq DB_2$ and $DB_2 \subseteq DB_1$ then $DB_1 \approx DB_2$?

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 $DB_1 \approx DB_2$

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

208

Computing a (Bi)Simulation

- Computing the maximal (bi)simulation:
  - R = {(s1,s2)| s1∈S1,s2∈S2,L1(s1)=L2(s2)}
  - 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 homomorphism** !
  - Compare to finding a **graph isomorphism** !

$\exists h((q,q')\in E \rightarrow (h(q),h(q'))\in E')$

NP-Complete

$\exists h((q,q')\in E \leftrightarrow (h(q),h(q'))\in E')$

NP-Hard ?

Kripke structure - bisimulation analysis

A symbolic version is also possible.
(skipped due to time-limit)

```
bisimulation(K,K') /* using greatest fixpoint algorithm */ {
   B={(s,s')| s∈S, s'∈S', L(s)=L(s') };
   repeat {
      for all (s,s') ∈B, {
         if ∃(s,t) ∈R, ∀(s',t') ∈R'((t,t') ∉ B), B = B - {(s,s')};
         if ∃(s',t') ∈R', ∀(s,t) ∈R((t,t') ∉ B), B = B - {(s,s')};
   } }  until no more changes to B for any (s,s').
   if ∃s0∈S0 ∀s0'∈S0' ((s0,s0') ∉ B), return "no bisimulation;"
   if ∃s0'∈S0' ∀s0∈S0 ((s0,s0') ∉ B), return "no bisimulation;"
   return "bisimulation exists."
}
```

The procedure terminates since $B \subseteq S \times S'$ is finite.

210

Language inclusion

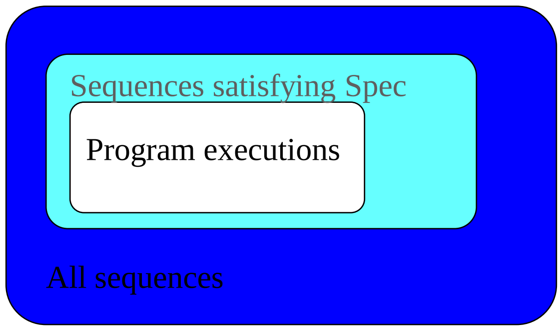
Since both can be modeled as automata, we can check the relation between their languages.

- Language of a model: L(Model).
- Language of a specification: L(Spec).

We need: L(Model) $\subseteq$ L(Spec).

211

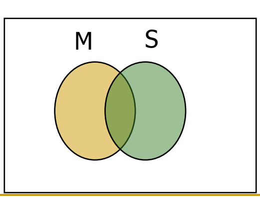
Language inclusion
- Correctness with runs



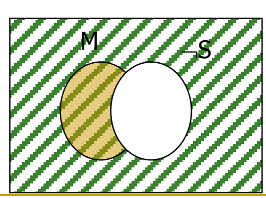
Language inclusion
- How to do it ?

- Show that L(Model) $\subseteq$ L(Spec).
- Equivalently:

  Show that L(Model) $\cap$ L($\neg$Spec) = Ø.
- How? Check that $A_{model} \cap A_{\neg Spec}$ is empty.



$M \subseteq S$



$M \cap \neg S \neq \emptyset$

Language inclusion
- What do we need to know?

$L(Model) \cap L(\neg Spec) = \emptyset$.

1. How to intersect two automata?
2. How to complement an automaton?
3. How to check for emptiness of an automaton ?
4. How to translate from LTL to an automaton ? (next week ...)

214

Language inclusion
- Automata for infinite sequences

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 Σ^ω
  - $\alpha = P_0, P_1, P_2, \ldots$ where $P_i = L(s_i)$.
- A word a is a model of formula f iff $\alpha \models f$
- Example: for $f = p \wedge (\neg q \; U \; q)$ $\{p\},\{\},\{q\},\{p,q\}^\omega$
- Let Mod(f) denote the set of models of f.

215

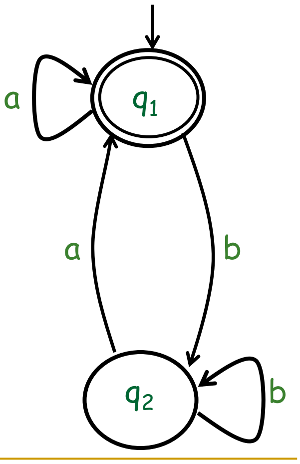
Language inclusion
- Büchi automata

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

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

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



Buchi Automaton

- Given an infinite word $w \in \Sigma^\omega$ where $w = a_0, a_1, a_2, \ldots$
  a run r of the automaton A over w is an infinite sequence of automaton states $r = q_0, q_1, q_2, \ldots$ where $q_0 \in I$ and for all $i \geq 0$, $(q_i,a_i,q_{i+1}) \in \Delta$
- Given a run r, let $inf(r) \subseteq Q$ be the set of automata states that appear in r infinitely many times
- A run r is an accepting run if and only if $inf(r) \cap F \neq \emptyset$
  i.e., a run is an accepting run if some accepting states appear in r infinitely many times

Transition System to Buchi Automaton Translation

Given a transition system $T = (S, I, R)$

a set of atomic propositions AP and

a labeling function $L : S \times AP \rightarrow \{true, false\}$

the corresponding Buchi automaton $A_T=(\Sigma_T, Q_T, \delta_T, I_T, F_T)$

$\Sigma_T = 2^{AP}$	an alphabet symbol corresponds to a set of atomic propositions
$Q_T = S \cup \{i\}$	i is a new state which is not in S
$I_T = \{i\}$	i is the only initial state
$F_T = S \cup \{i\}$	all states of A_T are accepting states

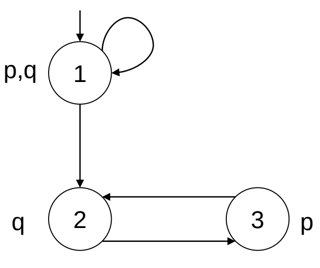
δ_T is defined as follows:

$(s,a,s') \in \delta_T$ iff either $(s,s') \in R$ and $L(s',a) = true$

or $s=i$ and $s' \in I$ and $L(s',a) = true$

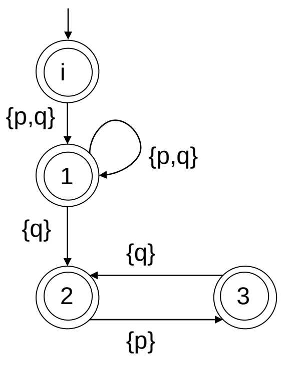
Transition System to Buchi Automaton Translation

Example transition system



Each state is labeled with the propositions that hold in that state

Corresponding Buchi automaton



Generalized Buchi Automaton

A generalized Buchi automaton is a tuple A = (Σ, Q, δ, I, F) where

Σ is a finite alphabet

Q is a finite set of states

$\delta \subseteq Q \times \Sigma \times Q$ is the transition relation

$I \subseteq Q$ is the set of initial states

$F \subseteq 2^Q$ is sets of accepting states

This is different than the standard definition

i.e., $F = \{F_1, F_2, \ldots, F_k\}$ where $F_i \subseteq Q$ for $1 \leq i \leq k$

- Given a generalized Buchi automaton A, a run r is an accepting run if and only if
  - for all $1 \leq i \leq k$, $inf(r) \cap F_i \neq \emptyset$

220

Buchi Automata Product

Given $A_1=(\Sigma,Q_1,\delta_1,I_1,F_1)$ and $A_2=(\Sigma,Q_2,\delta_2,I_2,F_2)$

the product automaton $A_1 \times A_2 = (\Sigma, Q, \delta, I, F)$ is defined as:

- $Q = Q_1 \times Q_2$
- $I = I_1 \times I_2$
- $F = \{F_1 \times Q_2, Q_1 \times F_2\}$ (a generalized Buchi automaton)
- δ is defined as follows:
- $((q_1,q_2),a,(q_1',q_2')) \in \delta$ iff $(q_1,a,q_1') \in \delta_1$ and $(q_2,a,q_2') \in \delta_2$

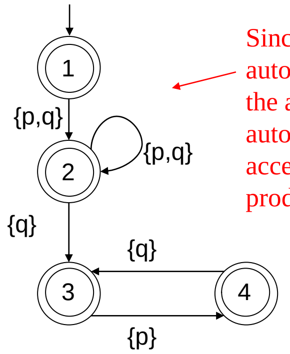
Based on the above construction, we get

$L(A_1 \times A_2) = L(A_1) \cap L(A_2)$

221

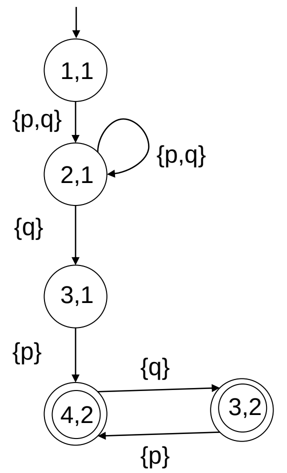
Example, a Special Case

Buchi automaton 1

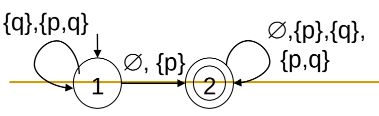


Since all the states in the automaton 1 is accepting, only the accepting states of automaton 2 decide the accepting states of the product automaton

Product automaton

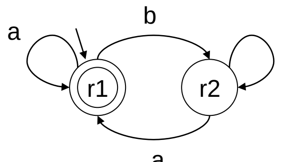


Buchi automaton 2



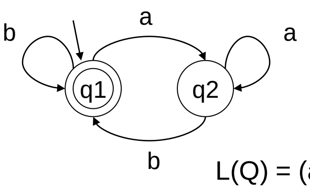
Buchi Automata Product Example

Automaton R



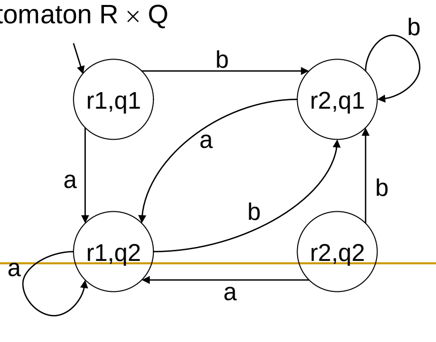
$L(R) = (b^*a)^\omega$

Automaton Q



$L(Q) = (a^*b)^\omega$

Automaton R × Q

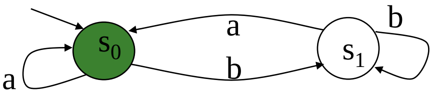


$L(R \times Q) = L(R) \cap L(Q)$

F = {
{(r1,q1), (r1,q2)},
{(r1,q1), (r2,q1)}
}

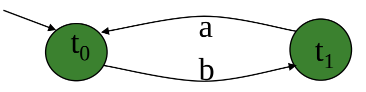
Language inclusion
- intersecting two finite-state automata

A_1:



$L(A_1) = (a+b)^*a + \varepsilon$
(words ending with 'a' + empty word)

A_2:

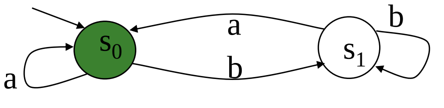


$L(A_2) = (ba)^* + (ba)^*b + \varepsilon$
(words that alternate between b and a + empty word)

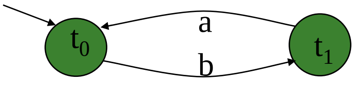
What should be the language of $A_1 \cap A_2$?

Language inclusion
- intersecting two finite-state automata

A_1:



A_2:

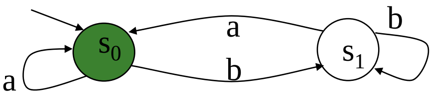


$A_1 \cap A_2$:

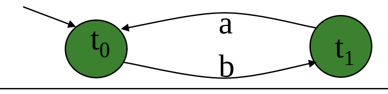
1. States: (s_0,t_0), (s_0,t_1), (s_1,t_0), (s_1,t_1).
2. Initial state(s): (s_0,t_0).
3. Accepting states: (s_0,t_0), (s_0,t_1).

Language inclusion
- intersecting two finite-state automata

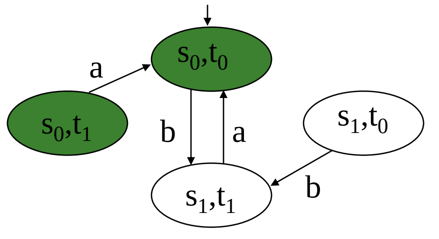
A_1:



A_2:



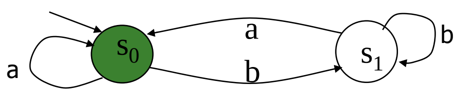
$A_1 \cap A_2$:



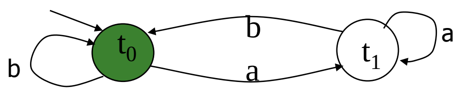
$L(A_1 \cap A_2) = (ba)^* + \varepsilon$

Language inclusion
- intersecting two Büchi automata

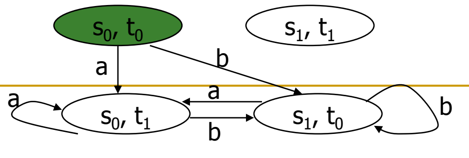
Previous method doesn't work:



Infinite a's



Infinite b's



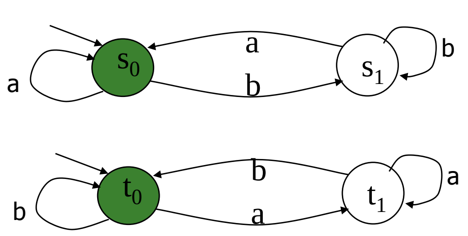
Empty language !

Language inclusion
- intersecting two Büchi automata

Strategy:

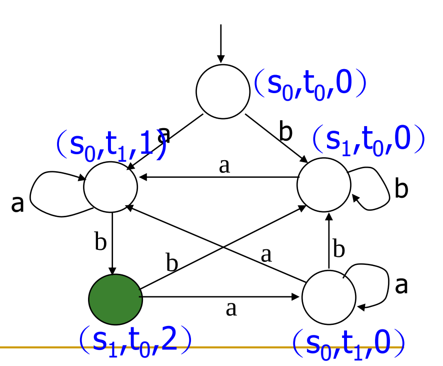
- "Multiply" the product automaton by 3 ($S = S1 \times S2 \times \{0,1,2\}$)
- Start from the '0' copy.
- Transition to the '1' copy when visiting a state from F1
- Transition to the '2' copy if in a '1' state and visiting a state from F2, and in the next state back to a '0' state.
- Make the '2' copy an accepting set.

Language inclusion
- intersecting two Büchi automata



There are total of 12 states in the product automaton.

The reachable part of $A_1 \cap A_2$ is:



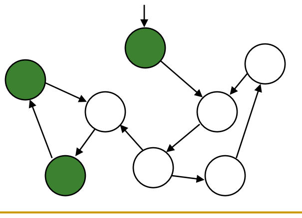
Language inclusion
- How to complement?

- Complementation is hard!
- We know how to translate an LTL formula to a Buchi automaton. So we can:
  - Build an automaton A for φ, and complement A, or
  - Negate the property, obtaining $\neg\varphi$ (the sequences that should never occur). Build an automaton for $\neg\varphi$.

230

Language inclusion
- How to check for emptiness?

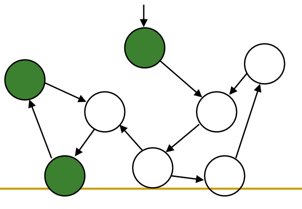
- Need to check if there exists an accepting run (passes through an accepting state infinitely often).
- This is called checking for emptiness, because if no such run exists, then L(A) = ;



231

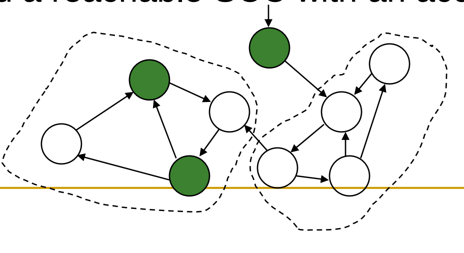
Language inclusion
- emptiness and accepting runs

- If there is an accepting run, then it contains at least one accepting state an infinite # of times.
- This state must appear in a cycle.
- So, find a reachable accepting state on a cycle.
- What graph algorithm ?



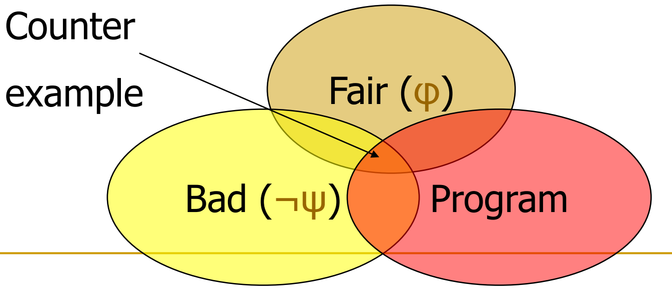
Language inclusion
- Finding accepting runs

- Rather than looking for cycles, look for SCCs:
  - A Strongly Connected Component (SCC): a set of nodes where each node is reachable from all others.
  - Finding SCC's is linear in the size of the graph.
  - Find a reachable SCC with an accepting node.



Language inclusion
- Verification under Fairness

Express the fairness as a property φ.
To prove a property ψ under fairness,
model check φ→ψ.



234