Computer Aided Verification
計算機輔助驗證

Model Checking (Part II)
模型檢驗 (二)

CTL, TCTL, and Symbolic Model Checking)
(Chapters: 4, 17, 5, 6 of “Model Checking” book)

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

These slide contents are adapted from the slides of Professor Farn Wang.

Contents

- CTL Model Checking
- Timed Automata (TA)
- Timed Computation Tree Logic (TCTL)
- TCTL Model Checking
- Clock Zones, DBM, BDD
- Symbolic Model Checking
- Counterexample & Witnesses

模型檢驗（Model Checking）

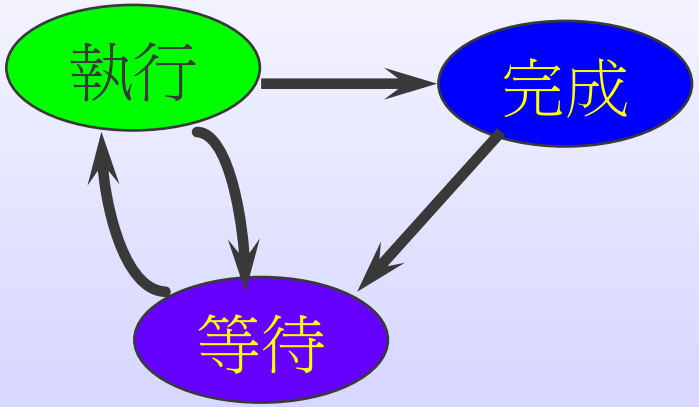
模型檢驗問題的定義：

給定一個有限狀態結構 M 與一個時態邏輯公式 φ，**請問M是 φ的一個模型嗎**？

$$\mathbf{M} \models \varphi\,?$$

模型檢驗（Model Checking）

- 狀態空間分析
  - 狀態空間表示法：*有向圖*
    - 節點：系統狀態
    - 箭頭：狀態轉換
- 規則系統行為



- 有限但極大(天文數字)之狀態空間

CTL模型檢驗的演算法

給定M 與 φ

1. 將$Closure(\varphi)$中的公式，由小到大排列

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

對所有$0 \le i < j \le n$， φ_j不是 φ_i的子公式！

2. 依序由$i=0$ 到 n，對所有M 中的狀態 s，計算 $M, s \models \varphi_i$ (第七頁的方法)

3. 如果M的啟始狀態是 s_0，且$M, s_0 \models \varphi$，則回答“是”;否則回答“否”!

Closure(φ) 是所有 φ 的子公式所成的集合

- $\neg\varphi$ 的子公式為 φ
- $\varphi\wedge\phi$ 、 $\varphi\vee\phi$ 的子公式為 φ 、 ϕ
- $\varphi\vee\phi$ 的子公式為 φ 、 ϕ
- $\exists\varphi\mathbf{U}\phi$ 的子公式為 φ 、 ϕ 、 $\exists\bigcirc\exists\varphi\mathbf{U}\phi$
- $\forall\varphi\mathbf{U}\phi$ 的子公式為 φ 、 ϕ 、 $\forall\bigcirc\forall\varphi\mathbf{U}\phi$
- $\exists\bigcirc\varphi$ 、 $\forall\bigcirc\varphi$ 的子公式為 φ
- $\exists\Diamond\varphi$ 的子公式為 φ 、 $\exists\bigcirc\exists\Diamond\varphi$
- $\exists\Box\varphi$ 的子公式為 φ 、 $\exists\bigcirc\exists\Box\varphi$
- $\forall\Diamond\varphi$ 的子公式為 φ 、 $\forall\bigcirc\forall\Diamond\varphi$
- $\forall\Box\varphi$ 的子公式為 φ 、 $\forall\bigcirc\forall\Box\varphi$

CTL模型檢驗的演算法（續）

- 若 φ_i 為 ***true***，則 $M,s \models \varphi_i$
- 若 φ_i 為 $\neg\varphi$，則 $M,s \models \varphi_i$ iff not $M,s \models \varphi$
- 若 φ_i 為 $\varphi \vee \phi$，則 $M,s \models \varphi_i$ iff $M,s \models \varphi$ 或 $M,s \models \phi$
- 若 φ_i 為 $\exists\varphi U \phi$，則由所有滿足 ϕ 的狀態開始，沿著滿足 φ 的所有的反向路徑上的狀態，都滿足 $\exists\varphi U \phi$
- 若 φ_i 為 $\exists\Box\varphi$，找出 M 中所有狀態都滿足 φ 的 ***Strongly Connected Components***（*SCC*）。由這些SCC開始，沿著滿足 φ 的所有的反向路徑上的狀態，都滿足 $\exists\Box\varphi$。
- <u>注意</u>： $\forall\varphi U \phi \equiv \neg((\exists\ \varphi\ U \neg(\varphi \vee \phi)) \vee \exists\Box\neg\phi)$

CTL模型檢驗的演算法（續）

- 若 φ_i為$\exists\varphi U\phi$，則由所有滿足ϕ的狀態開始，沿著滿足φ的所有的反向路徑上的狀態，都滿足$\exists\varphi U\phi$。

1 For all **state *s* with *M,s*** $\models\phi$**, *M,s*** $\models\exists\varphi U\phi$ holds

2 Repeat **until no more change can be made:**

3 For all **(*s'*, *s*),** if ***M,s'*** $\models\varphi$ and ***M,s*** $\models\exists\varphi U\phi$**,**

4 then ***M,s'*** $\models\exists\varphi U\phi$

*此一計算一定會停止，因為每一個$\exists\varphi U\phi$都是被有限長度路徑所滿足。**Repeat**迴圈只需循環到最長的最短路徑長度。*

CheckEU(f_1, f_2) for $\exists\, f_1 U f_2$

```
procedure CheckEU(f1, f2)
    T := { s | f2 ∈ label(s) };
    for all s ∈ T do label(s) := label(s) ∪ { E[f1 U f2] };
    while T ≠ ∅ do
        choose s ∈ T;
        T := T \ {s};
        for all t such that R(t, s) do
            if E[f1 U f2] ∉ label(t) and f1 ∈ label(t) then
                label(t) := label(t) ∪ { E[f1 U f2] };
                T := T ∪ {t};
            end if;
        end for all;
    end while;
end procedure
```

Figure 4.1
Procedure for labeling the states satisfying $\mathbf{E}[f_1\ \mathbf{U}\ f_2]$.

CTL模型檢驗的演算法（續）

- 若 φ_i 為 $\exists\Box\varphi$，找出 M 中所有狀態都滿足 φ 的*Strongly Connected Components*（*SCC*）。由這些SCC開始，沿著滿足 φ 的所有的反向路徑上的狀態，都滿足 $\exists\Box\varphi$。

1 **將M中，不滿足 φ 的狀態都刪掉。**

2 **執行transitive closure以計算剩餘狀態間的路徑。**

Reach[*s*,*s'*]=1如果由*s*到*s'*之間，有一條路徑

Reach[*s*,*s'*]=0如果由*s*到*s'*之間，沒有路徑

3 For all ***s*, *M*,*s*** $\models \exists\Box\varphi$ iff $\exists$***s'*(Reach[*s*,*s'*]** $\wedge$ **Reach[*s'*,*s'*])**

CTL模型檢驗的演算法（續）

在M'（剩餘的M）中，所有狀態都滿足φ。

<u>執行 transitive closure:</u>

1 For all s, s' do

if $(s, s') \in M'$, **Reach[s, s'] := 1;**

else **Reach[s, s'] := 0;**

2 For s, s' do

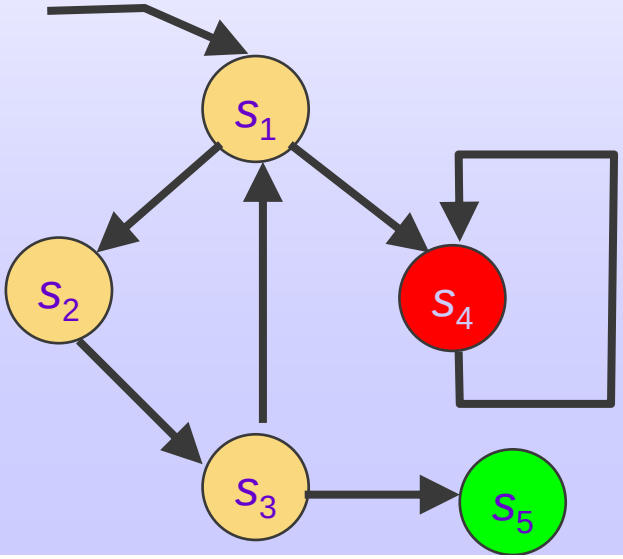
For all s'' do

Reach[s, s'] := Reach[s, s'] $\vee$

(Reach[s, s''] $\wedge$ Reach[s'', s']);

Strongly Connected Components (SCC)

- A strongly connected component (SCC) *C* is a maximal sub-graph such that every node in *C* is reachable from every other node in *C* along a directed path entirely contained within *C*.



Two SCC:

$\{s_1, s_2, s_3\}$

$\{s_4\}$

CheckEG(f_1) for $\exists\Box f_1$

```
procedure CheckEG(f1)
    S' := { s | f1 ∈ label(s) };
    SCC := { C | C is a nontrivial SCC of S' };
    T := ⋃_{C∈SCC} { s | s ∈ C };
    for all s ∈ T do label(s) := label(s) ∪ { EG f1 };
    while T ≠ ∅ do
        choose s ∈ T;
        T := T \ {s};
        for all t such that t ∈ S' and R(t, s) do
            if EG f1 ∉ label(t) then
                label(t) := label(t) ∪ { EG f1 };
                T := T ∪ {t};
            end if;
        end for all;
    end while;
end procedure
```

Figure 4.2
Procedure for labeling the states satisfying EG f_1.

Microwave Oven Example

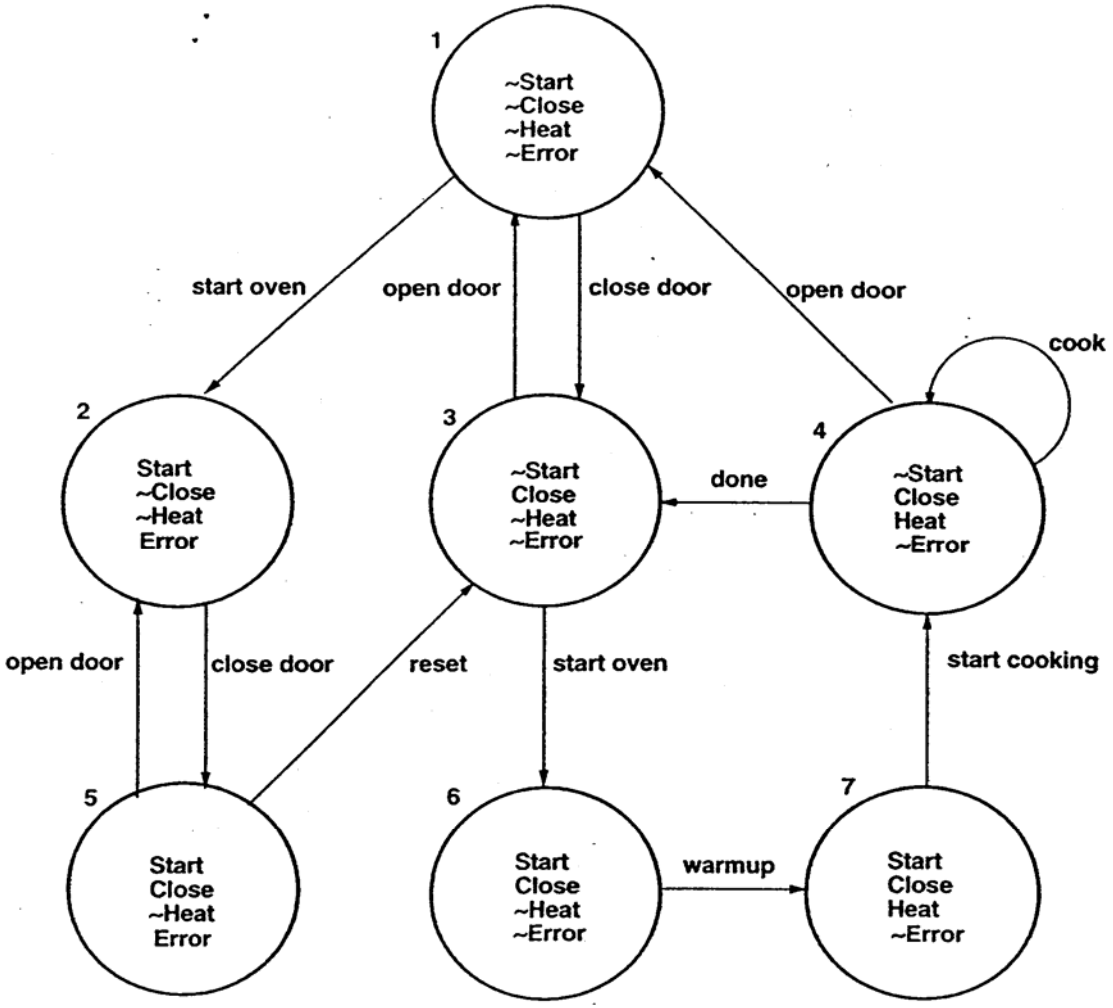


Figure 4.3
Microwave oven example.

Microwave Oven Example

- $\forall\Box(\text{Start} \rightarrow \forall\Diamond\text{Heat})$
- $\neg\exists\Diamond(\text{Start} \wedge \exists\Box\neg\text{Heat})$
- $S(\text{Start}) = \{2, 5, 6, 7\}$
- $S(\neg\text{Heat}) = \{1, 2, 3, 5, 6\}$
- $SCC(\neg\text{Heat}) = \{1, 2, 3, 5\}$
- $S(\exists\Box\neg\text{Heat}) = \{1, 2, 3, 5\}$
- $S(\text{Start} \wedge \exists\Box\neg\text{Heat}) = \{2, 5\}$
- $S(\exists\Diamond(\text{Start} \wedge \exists\Box\neg\text{Heat})) = \{1,2,3,4,5,6,7\}$
- $S(\neg\exists\Diamond(\text{Start} \wedge \exists\Box\neg\text{Heat})) = \varnothing$

Fairness Constraints

- $F = \{P_1, \ldots, P_k\}$ // fairness constraints
- Fair SCC $C \Leftrightarrow \forall P_i \in F, \exists$state $t_i \in (C \cap P_i)$.
- CheckFairEG(f_1): Fair SCCs
- Define "*fair*": atomic proposition, true at a state s iff there is a fair path starting at s.
- $M, s \models p \wedge fair$, $M, s \models \exists \bigcirc (f_1 \wedge fair)$,
- CheckEUFair(f_1, f_2) = CheckEU($f_1, f_2 \wedge fair$) $= M, s \models \exists (f_1 \mathrm{U} (f_2 \wedge fair))$

Microwave Oven (with Fairness)

- $\forall\Box(\text{Start} \rightarrow \forall\Diamond\text{Heat})$
- $F = \{\Box\Diamond(\text{Start} \wedge \text{Close} \wedge \neg\text{Error})\}$
- $S(\text{Start}) = \{2, 5, 6, 7\}$
- $S(\neg\text{Heat}) = \{1, 2, 3, 5, 6\}$
- $SCC(\neg\text{Heat}) = \{1, 2, 3, 5\}$ is NOT fair
- $S(\exists\Box\neg\text{Heat}) = \{\ \}$
- $S(\exists\Diamond(\text{Start} \wedge \exists\Box\neg\text{Heat})) = \{\ \}$
- $S(\neg\exists\Diamond(\text{Start} \wedge \exists\Box\neg\text{Heat}))=\{1,2,3,4,5,6,7\}$

Model Checking Real-Time Systems

- Real-Time Systems:
  - Timed Automata (TA)
- Real-Time Properties:
  - Timed Computation Tree Logic (TCTL)
- **Given a real-time system *S* modeled by a set of TA $\{A_1, A_2, \ldots, A_n\}$ and a real-time property specified by a TCTL formula ϕ, check if *S* satisfies ϕ ($S \models \phi$?).**

Timed Automata真時自動機

<u>狀態條件敘述（state predicate）</u>：

以**proposition集合P**與**時鐘變數集合X**，建構而成

$$\eta ::= p \mid x \sim c \mid x\text{-}y \sim y \mid \neg\eta_1 \mid \eta_1 \vee \eta_2$$

$$p \in P;\ x, y \in X;\ c, d \in N;\ \text{‘}\sim\text{’} \in \{\leq, \geq, <, >, =\}$$

<u>範例</u>：

男性 ∧ 未婚 ∧ 年齡 ≤ 30

<u>範例</u>：

孤單 ∧ 男性 ∧ 未婚 ∧ 今日+2 ≤ 收假日

<u>範例</u>：

戰機巡航模式 ∧ x+5 < 回報週期

Timed Automata真時自動機

<u>狀態條件敘述（state predicate）</u>：

以**proposition**集合**P**與時鐘變數集合**X**，建構而成

$$\eta ::= p \mid x \sim c \mid x+c \sim y+d \mid \neg\eta_1 \mid \eta_1 \vee \eta_2$$

$$p \in P;\ x, y \in X;\ c, d \in N;\ \text{‘}\sim\text{’} \in \{\leq, \geq, <, >, =\}$$

<u>縮寫</u>：

$$\begin{aligned} true &\equiv x = x \\ \eta_1 \wedge \eta_2 &\equiv \neg((\neg\eta_1) \vee (\neg\eta_2)) \\ \eta_1 \rightarrow \eta_2 &\equiv (\neg\eta_1) \vee \eta_2 \end{aligned}$$

<u>$\Gamma(P, X)$</u>：

所有由P、X建構而成的狀態條件敘述之集合

Timed Automata真時自動機

$A = \langle Q, q_0, P, X, \mu, E, \tau, \pi \rangle$

Q : 有限控制點的集合 (set of control locations)

q_0 : 啟始控制點 (initial location)

P : 命題的集合 (set of propositions)

X : 時鐘變數的集合 (set of clock variables)

$\mu : Q \rightarrow \Gamma(P,X)$；各控制點的恆定條件 (invariant)

$E \subseteq Q \times Q$: 控制點轉換的集合 (set of transitions)

$\tau : E \rightarrow \Gamma(P, X)$；各轉換的激發條件 (triggering condition)

$\pi : E \rightarrow 2^X$: 在轉換時，要被reset成零（歸零）的時鐘變數的集合 (set of clocks to be reset)

$A=\langle Q, q_0, P, X, \mu, E, \tau, \pi\rangle$

Q={監控，命中}

E={(監控,監控),(監控,命中)}

q_0=監控

τ(監控,監控)= z =50

P = { }, X = {x, y}

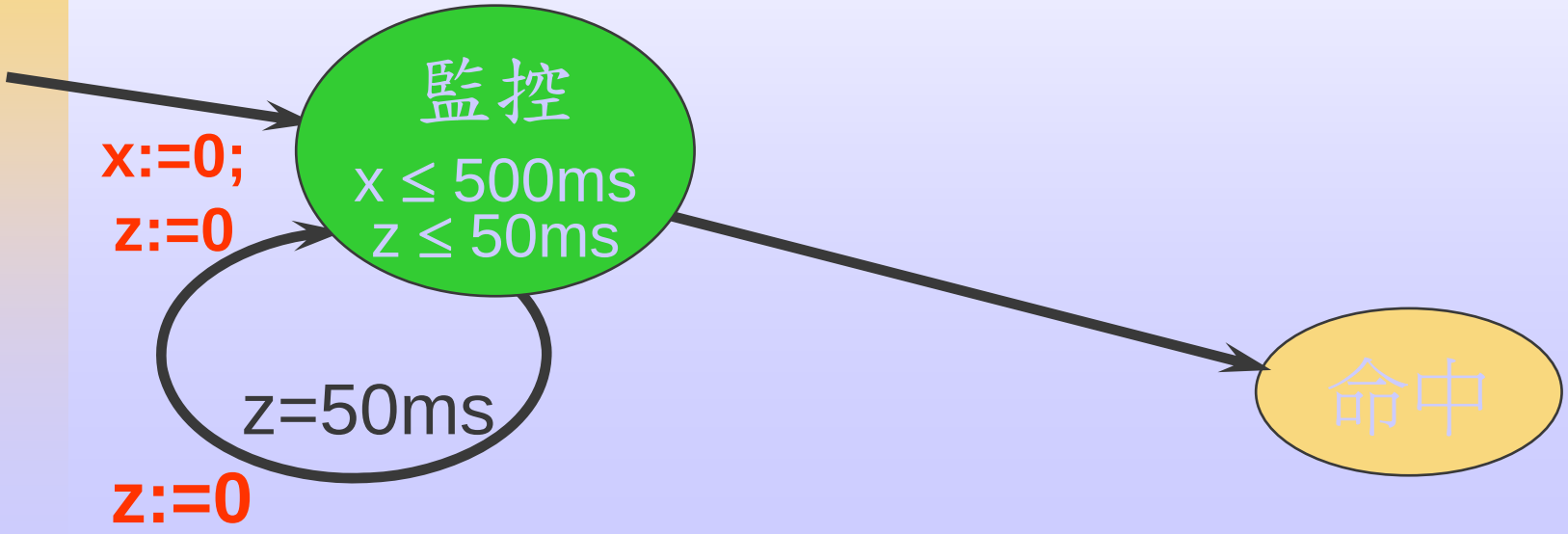
τ(監控,命中)= *true*

μ(監控)= $x \le 500 \wedge z \le 50$

π(監控,監控)= {z}

μ(命中)= *true*

π(監控,命中)= {}



Parallel Composition of TA

- $A_1 = \langle \boldsymbol{Q_1}, \boldsymbol{q_{01}}, \boldsymbol{P_1}, \boldsymbol{X_1}, \boldsymbol{\mu_1}, \boldsymbol{E_1}, \boldsymbol{\tau_1}, \boldsymbol{\pi_1} \rangle$,
  $A_2 = \langle \boldsymbol{Q_2}, \boldsymbol{q_{02}}, \boldsymbol{P_2}, \boldsymbol{X_2}, \boldsymbol{\mu_2}, \boldsymbol{E_2}, \boldsymbol{\tau_2}, \boldsymbol{\pi_2} \rangle$
- $\boldsymbol{A_1} \parallel \boldsymbol{A_2} = \langle \boldsymbol{Q_1} \times \boldsymbol{Q_2}, \boldsymbol{q_0}, \boldsymbol{P_1} \cup \boldsymbol{P_2}\ \boldsymbol{X_1} \cup \boldsymbol{X_2}\ \boldsymbol{\mu},$
  $\boldsymbol{E}, \boldsymbol{\tau_1} \wedge \boldsymbol{\tau_2}, \boldsymbol{\pi_1} \cup \boldsymbol{\pi_2} \rangle$
- q_0: a merge of q_{01} and q_{02}
- $\mu(q, q') = \mu_1(q) \wedge \mu_2(q')$
- E: set of interleaved transitions with synchronization transitions identified

A Manufacturing Example

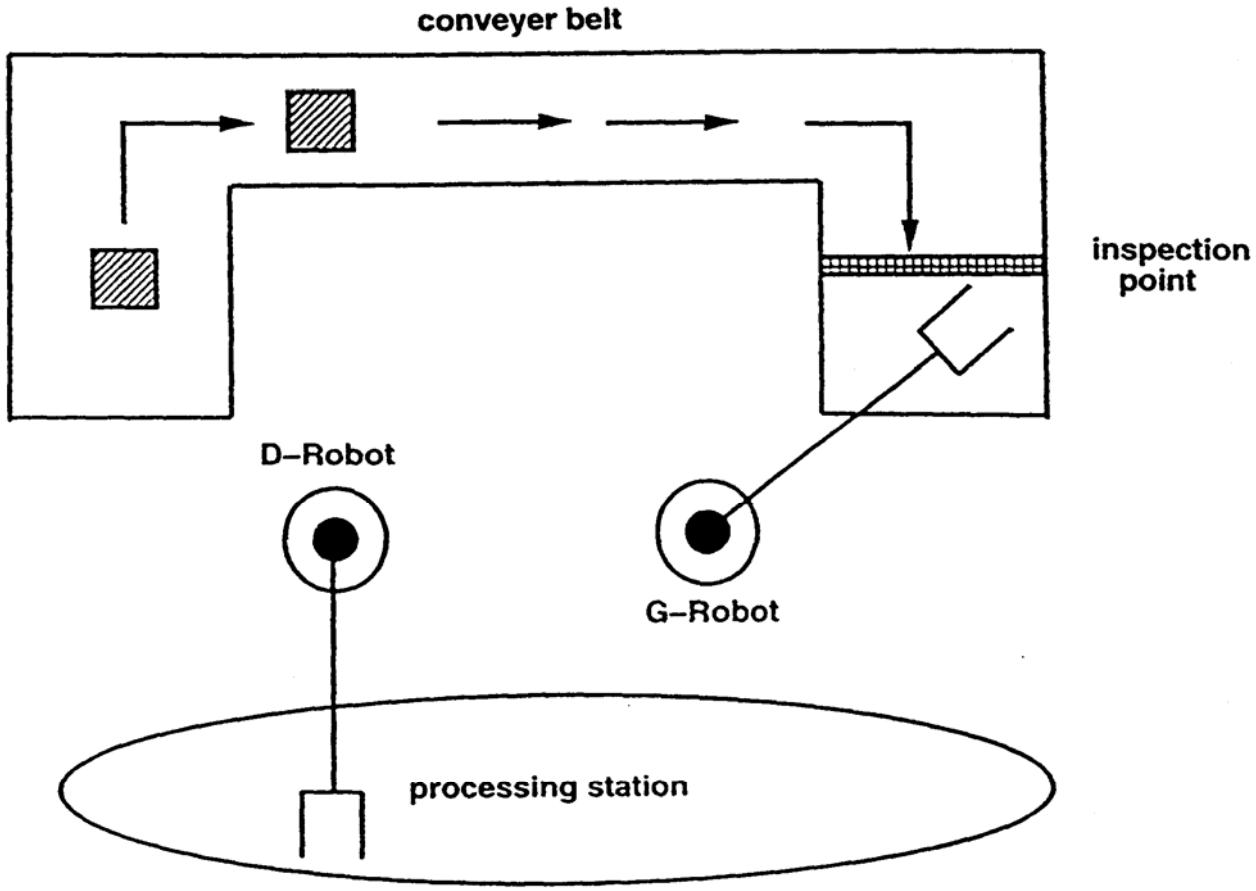


Figure 17.2
A manufacturing example.

Manufacturing Ex: D-Robot TA

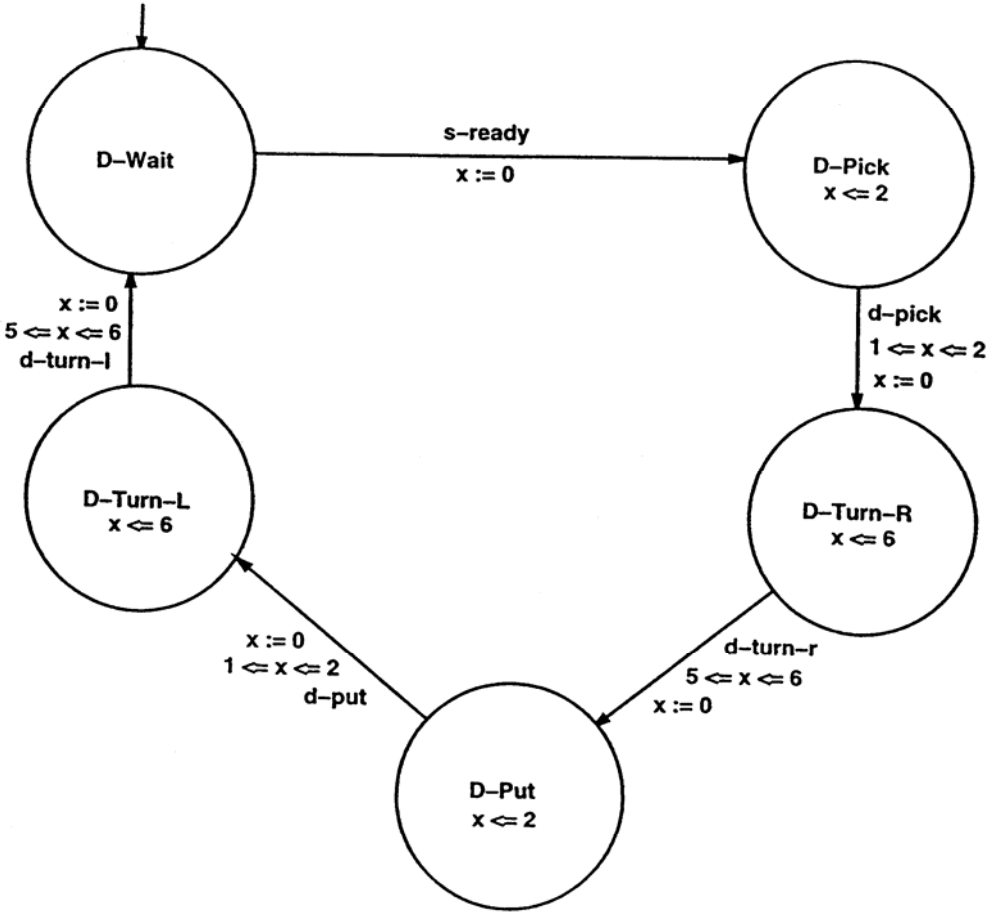


Figure 17.3
Timed automaton for D-Robot.

Manufacturing Ex: G-Robot TA

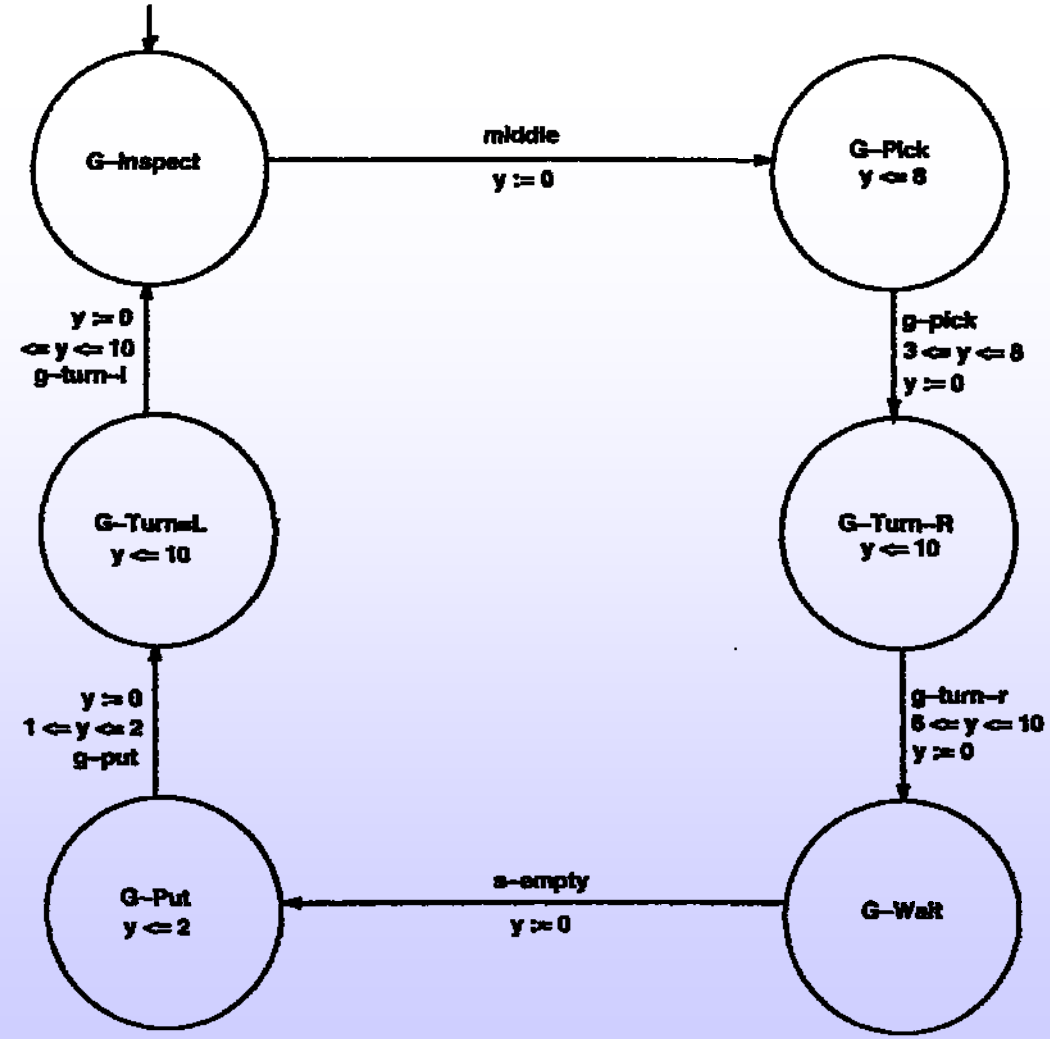


Figure 17.4
Timed automaton for G-Robot.

Manufacturing Ex: Station TA

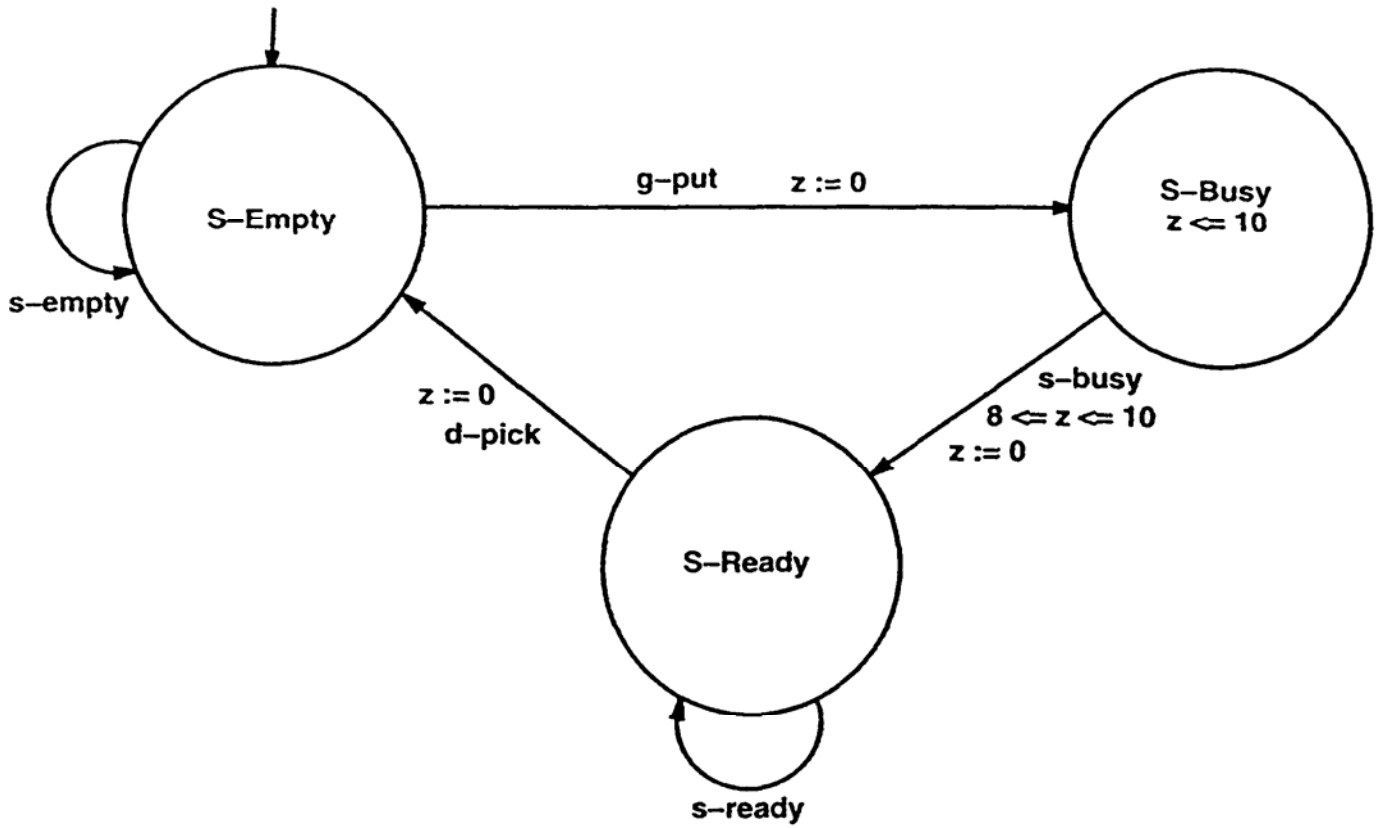


Figure 17.5
Time automaton for processing station.

Manufacturing Ex: Box TA

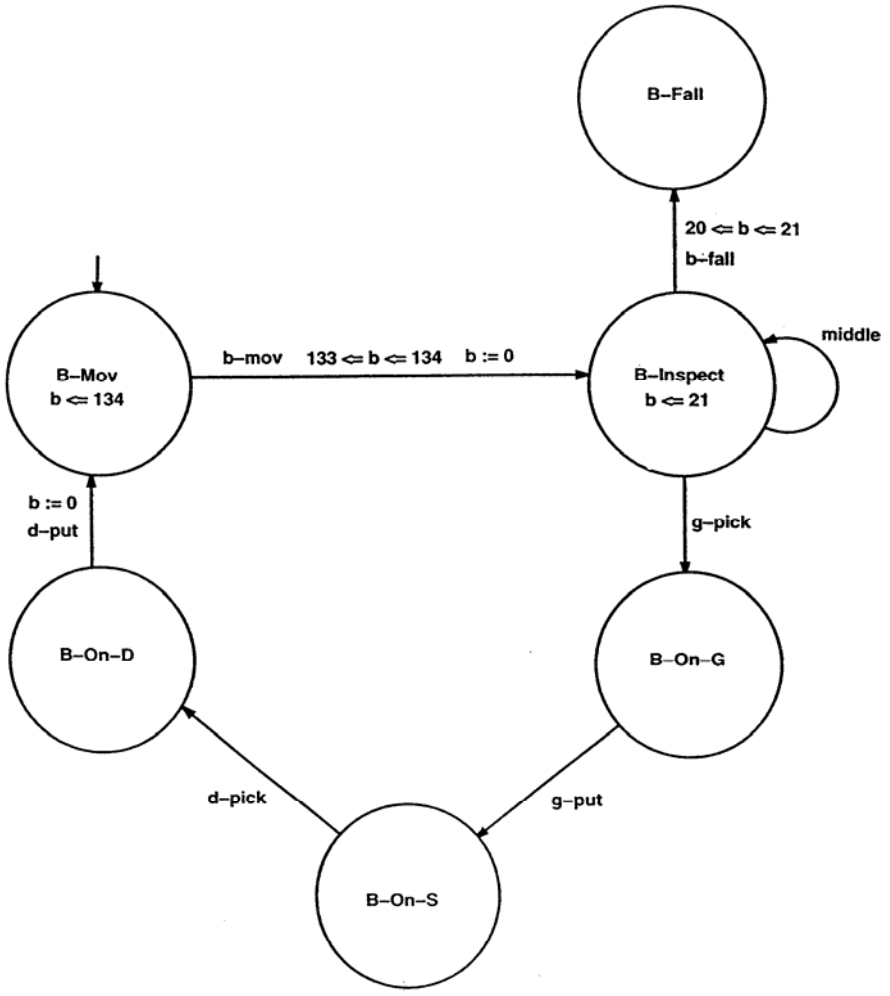


Figure 17.6
Timed automaton for box.

Model of Manufacturing System

- M = D-Robot || G-Robot || Station || Box
- Transitions with same labels are identified as one in M.
- E.g.: g-put in G-Robot, Station, and Box
- E.g.: d-pick in D-Robot, Station, and Box

真時自動機的語意

$A = \langle Q, q_0, P, X, \mu, E, \tau, \pi \rangle$

<u>狀態</u> $s = (q, \nu)$

- $q \in Q$
- $\nu : P \rightarrow \{true, false\} \cup X \rightarrow R^+$;
  - 定義所有Boolean變數的真假值
  - 定義所有時鐘變數的非負實數讀值
  - $\nu \models \mu(q)$;
    - 時鐘讀值與proposition設定，須滿足控制點的恆定條件

真時自動機的語意

$A=\langle Q, q_0, P, X, \mu, E, \tau, \pi\rangle$

$\nu \models \eta$：ν 滿足一個狀態條件敘述

(state predicate) η

$\nu \models p$	iff	$\nu(p) = true$
$\nu \models x \sim c$	iff	$\nu(x) \sim c$
$\nu \models x - y \sim d$	iff	$\nu(x) - \nu(y) \sim d$
$\nu \models \neg\eta_1$	iff	not $\nu \models \eta_1$
$\nu \models \eta_1 \vee \eta_2$	iff	$\nu \models \eta_1$ 或 $\nu \models \eta_2$

<u>狀態</u> $s = (q, \nu)$ ； $s \models \eta$ iff $\nu \models \eta$

真時自動機的語意

兩個狀態的符號定義

- 給定一個$\delta \in R^+$， $\nu + \delta$是一個新函數

$$(\nu + \delta)(p) = \nu(p)$$

$$(\nu + \delta)(x) = \nu(x) + \delta$$

- 給定一個 $Y \subseteq X$ ， νY是一個新函數

$$(\nu Y)(p) = \nu(p)$$

$$(\nu Y)(x) = 0 \qquad \text{iff } x \in Y$$

$$(\nu Y)(x) = \nu(x) \quad \text{iff } x \notin Y$$

- 給定 $s = (q, \nu)$ 與 $Y \subseteq X$ ， $sY = (q, \nu Y)$

真時自動機的語意

從直觀上看，

dense-time自動機的計算（computation），應該是

- 對映到R^+（非負實數集合）的一個狀態序列；
- 具有狀態的稠密性；

真時自動機的語意

$A=\langle Q, q_0, P, X, \mu, E, \tau, \pi\rangle$

一個轉換 (transition) 的符號定義

給定$s = (q,\nu)$ 與 $s' = (q',\nu')$ ，

$s \rightarrow s'$（狀態 s與狀態 s'間，有一個轉換）iff

- $(q, q') \in E$
- $s \models \tau(q, q')$
- $s\pi(q, q') \models \mu(q')$

真時自動機的語意

$A=\langle Q, q_0, P, X, \mu, E, \tau, \pi\rangle$ 的計算 (computation)

s_0-run: $s_0\, s_1\, s_2 \ldots s_k \ldots \ldots \ldots$

- 對所有$k \geq 0$， s_k是具有如(q,ν)格式的狀態。
- 存在一個monotonically increasing、divergent的非負實數序列

$$t_0\, t_1\, t_2 \ldots t_k \ldots \ldots \ldots$$

對所有$k \geq 0$，

- 對所有 $t \in [0, t_{k+1}-t_k]$， $s_k + t \models \mu(q)$
- Either $s_k + (t_{k+1}-t_k) = s_{k+1}$ or $s_k + (t_{k+1}-t_k) \rightarrow s_{k+1}$

Timed Computation Tree Logic (TCTL)

<u>範例</u>：從現在的狀態開始，有一種可能將來，七天會發薪水。

$$\exists\Diamond_{=7}\text{發薪水}$$

<u>範例</u>：從現在開始，無論將來如何，十年內要結婚。

$$\forall\Diamond_{\leq 10}\text{結婚}$$

TCTL（續）

<u>範例</u>：在任何計算狀態，敵機出現，都可能在五秒內擊落。

$$\forall\Box(\text{發現敵機} \rightarrow \exists\Diamond_{\leq 5}\ \text{擊落敵機})$$

<u>範例</u>： 在任何狀態，不管如何計算，敵機都會在五秒內擊落。

$$\forall\Box(\text{發現敵機} \rightarrow \forall\Diamond_{\leq 5}\ \text{擊落敵機})$$

TCTL（續）

<u>範例</u>：從任何狀態開始，都有一種計算，時間會無限增加。

$$\forall\Box\exists\Diamond_{=1}\mathit{true}$$

<u>範例</u>：從任何狀態開始，對所有計算，時間都會無限增加。

$$\forall\Box\forall\Diamond_{=1}\mathit{true}$$

TCTL的語法

$$\varphi ::= \eta \mid \neg \varphi_1 \mid \varphi_1 \vee \varphi_2 \mid \exists \varphi_1 U_{\sim c} \varphi_2 \mid \exists \Box_{\sim c} \varphi_1$$

$\eta \in \Gamma(P, X)$：所有由P、X建構而成的狀態條件敘述之集合;

$c \in N$;

'~' $\in \{\leq, \geq, <, >, =\}$

TCTL的語法

<u>縮寫</u>：

$$\varphi_1 \wedge \varphi_2 \equiv \neg((\neg \varphi_1) \vee (\neg \varphi_2))$$

$$\varphi_1 \rightarrow \varphi_2 \equiv (\neg \varphi_1) \vee \varphi_2$$

$$\exists \Diamond_{\sim c} \varphi \equiv \exists\, \mathit{true}\, U_{\sim c}\, \varphi$$

$$\forall \Box_{\sim c} \varphi \equiv \neg \exists \Diamond_{\sim c} \neg \varphi$$

$$\forall\, \varphi_1 U_{\sim c} \varphi_2 \equiv \neg((\exists (\neg \varphi_2) U_{\sim c} \neg(\varphi_1 \vee \varphi_2)) \vee \exists \Box_{\sim c} \neg \varphi_2)$$

$$\forall \Diamond_{\sim c} \varphi \equiv \forall\, \mathit{true}\, U_{\sim c}\, \varphi$$

TCTL的語意

$A=\langle Q, q_0, P, X, \mu, E, \tau, \pi\rangle$

$A,(q,v) \models \varphi$; 狀態 (q,v) 滿足 φ

$A,(q,v) \models \eta$； 定義如state predicate 者

$A,(q,v) \models \neg \varphi_1$ iff not $A,(q,v) \models \varphi_1$

$A,(q,v) \models \varphi_1 \vee \varphi_2$ iff $A,(q,v) \models \varphi_1$ 或 $A,(q,v) \models \varphi_2$

$A,(q,v) \models \exists \varphi_1 U_{\sim c} \varphi_2$ iff ...

$A,(q,v) \models \exists \Box_{\sim c} \varphi_1$ iff ...

TCTL的語意

$A=\langle Q, q_0, P, X, \mu, E, \tau, \pi\rangle$

$A,(q,v) \models \exists\, \varphi_1 U_{\sim c}\, \varphi_2$ iff 存在

- (q,v)-run：$s_0\, s_1\, s_2 \ldots s_k \ldots \ldots \ldots$ ，與
- monotonically increasing、divergent的非負實數序列：$t_0\, t_1\, t_2 \ldots\ldots$與
- $k \geq 0$，與 $t \in [0, t_{k+1} - t_k]$，
  - $t_k + t - t_0 \sim c$
  - $A, s_k + t \models \varphi_2$
  - 對所有$k > h \geq 0$與 $t' \in [0, t_{h+1} - t_h]$， $A, s_h + t' \models \varphi_1$
  - 對所有 $t' \in [0, t)$，$A, s_k + t' \models \varphi_1$

TCTL的語意

$A = \langle Q, q_0, P, X, \mu, E, \tau, \pi \rangle$

$A,(q,v) \models \exists\Box_{\sim c}\, \varphi_1$ iff 存在

- (q,v)-run：$s_0\, s_1\, s_2 \ldots s_k \ldots \ldots \ldots$ ，與
- monotonically increasing、divergent的非負實數序列：$t_0\, t_1\, t_2 \ldots\ldots$

對所有$k \geq 0$，與 $t \in [0, t_{k+1} - t_k]$，

若 $t_k + t - t_0 \sim c$，則 $A, s_k + t \models \varphi_1$

TCTL模型檢驗問題

<u>定義</u>：$A \models \varphi$（A滿足 φ ；A是 φ 的一個模型）

給定一個$A=\langle Q, q_0, \mu, E, \tau, \pi\rangle$與一個TCTL公式 φ，

$A \models \varphi$ iff $\forall v$， 若 $v \models \mu(q_0)$，則$A,(q_0, v) \models \varphi$

<u>TCTL模型檢驗問題</u>：

給定一個$A=\langle Q, q_0, \mu, E, \tau, \pi\rangle$與一個TCTL公式 φ，

請問A是 φ 的一個模型嗎？

TCTL模型檢驗問題的反思

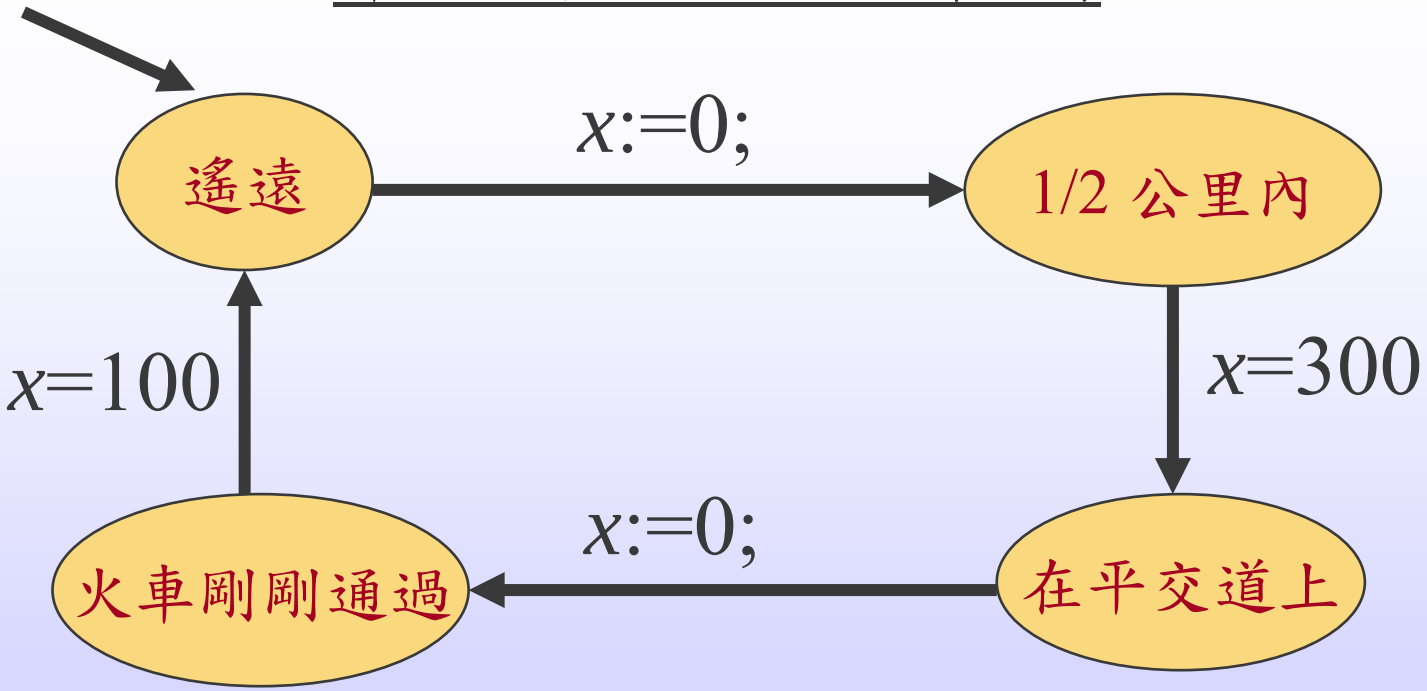
如何在無限、稠密的狀態空間中，分割出有限的discrete狀態空間？

工程經驗：

在任何即時系統中，雖然面對無限的大自然環境，在電腦的觀測中，還是以有限的flag、switch來描述，並控制系統的行為。

TCTL模型檢驗問題的反思

平交道監視器的範例



在即時系統中，用clock來控制!

TCTL模型檢驗問題的反思

- *Real-time systems 的自動驗證，真的是必要的嗎？*
- *還是只是學術上的假象？*

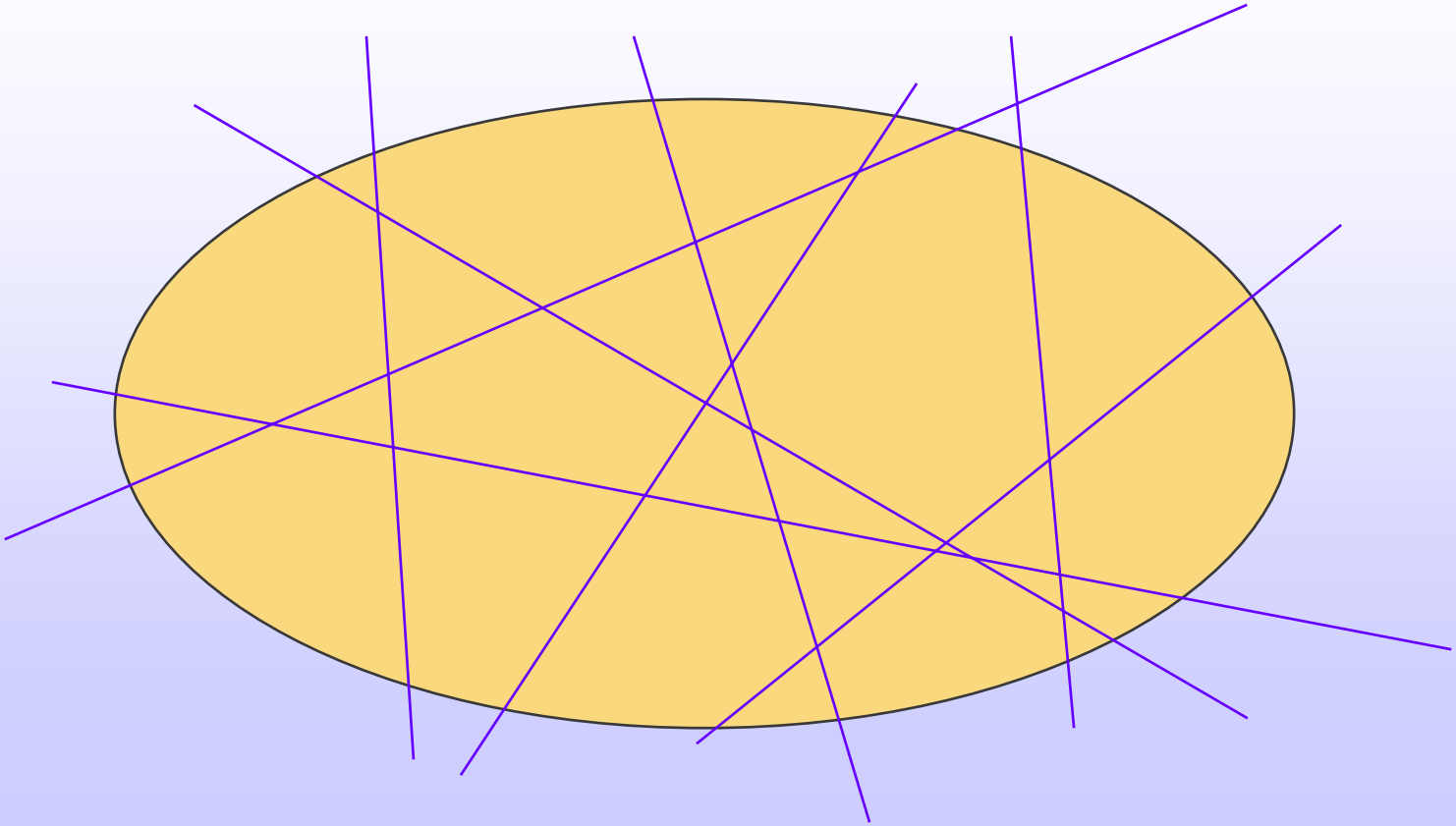
目前的一個困境：

- 在即時系統的製作上，工業界的工程師並不把系統看做**real-time automata**
  - 最傳統的作法，還是**periodic model**
- 在真時工業案例中，也缺乏強有力的理由，非用**real-time automata**不可。

TCTL模型檢驗問題

如何將無限的狀態空間，

切割為有限、且等價行為的狀態子空間？



TCTL模型檢驗問題

符號定義：

假設在$\boldsymbol{A}$與φ中，使用最大的常數是

$C_{\boldsymbol{A}:\varphi}$。

TCTL模型檢驗問題

Infinite States → Finite Regions

Two states are in the same region, i.e.,

$(q, \nu) \equiv (q', \nu')$，若

- $q = q'$
- 對所有 $p \in P$，$\nu(p) = \nu'(p)$
- 對所有 $x \in X$，$\nu(x)$ 與 $\nu'(x)$ 在小於 $C_{A:\varphi}$ 時，整數部分相等。

  若 $\nu(x) \le C \wedge \nu'(x) \le C$ ，則 $\lfloor \nu(x) \rfloor = \lfloor \nu'(x) \rfloor$
- 對所有 $x, y \in X$，$\nu(x)$ 與 $\nu'(x)$ 小於 $C_{A:\varphi}$ 時，小數部分的順序相同。

  若 $\nu(x) \le C \wedge \nu'(x) \le C$ ，則

$$\nu(x) - \lfloor \nu(x) \rfloor \le \nu(y) - \lfloor \nu(y) \rfloor \quad \text{iff}$$
$$\nu'(x) - \lfloor \nu'(x) \rfloor \le \nu'(y) - \lfloor \nu'(y) \rfloor$$

TCTL模型檢驗問題

當$C_{A:\varphi}$=7 且有兩個clock: x、y

(q, v)	(q, v')	
$v(x)=3.5$ $v(y)=5.5$	$v'(x)=3.7$ $v'(y)=5.3$	不等價
$v(x)=3.2$ $v(y)=5.5$	$v'(x)=4.3$ $v'(y)=5.8$	不等價
$v(x)=3.1$ $v(y)=5.8$	$v'(x)=3.5$ $v'(y)=5.51$	等價
$v(x)=13.5$ $v(y)= 8.5$	$v'(x)= 89$ $v'(y)=1003$	等價

2 Clocks with $c_x = 2$, $c_y = 1$

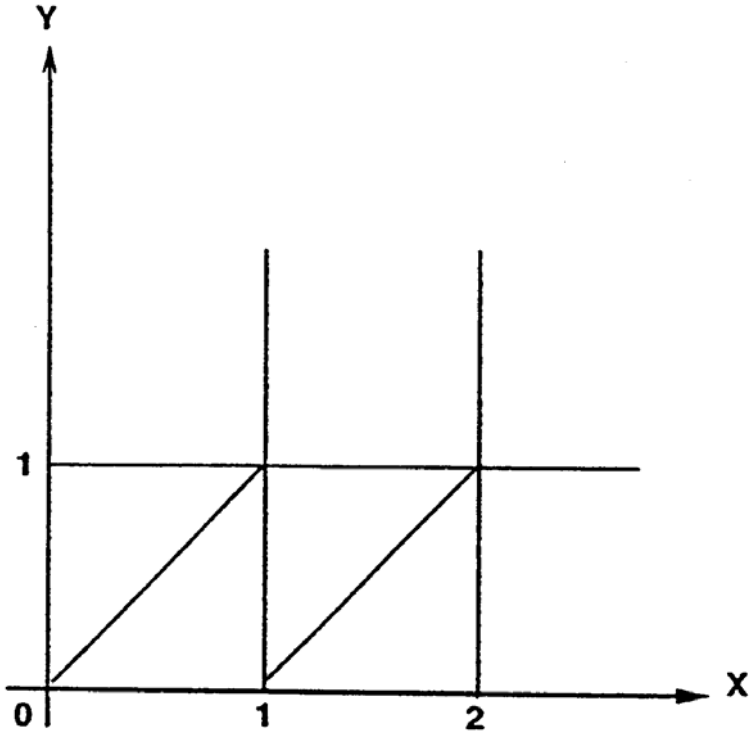


Figure 17.7
Clock region example.

28 Clock Regions:

- **6 corner points,**
- **14 open line segments,**
- **8 open regions**

TCTL模型檢驗問題

Equivalent States: $(q, \nu) \equiv (q', \nu')$，若

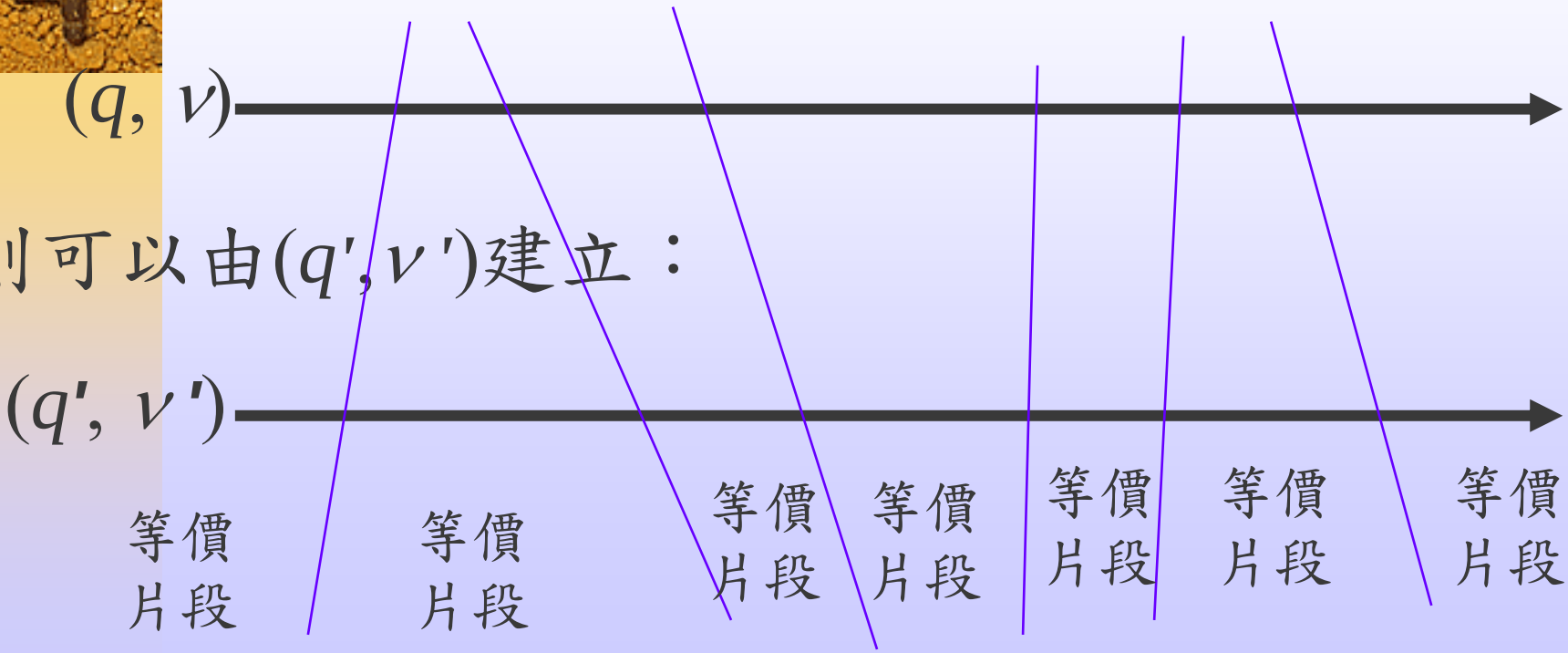
- $q = q'$
  - 立於相同的控制點。
- 對所有 $p \in P$，$\nu(p) = \nu'(p)$
  - 可以判斷所有 **Boolean** 變數的真假值。
- 對所有$x \in X$，$\nu(x)$ 與$\nu'(x)$ 在小於$C_{A:\varphi}$時，整數部分相等
  - 可以判斷所有 $x \sim c$不等式的真假值。
- 對所有$x, y \in X$，$\nu(x)$ 與$\nu'(x)$ 在小於$C_{A:\varphi}$時，小數部分的順序相同。
  - 可以判斷哪一個不等式 $x \sim c$會先發生變化。

TCTL模型檢驗問題

定理：對所有 ϕ，若$(q, \nu) \equiv (q', \nu')$，則

$$A, (q, \nu) \models \phi \text{ iff } A, (q', \nu') \models \phi$$

直觀證明：若存在



TCTL模型檢驗問題

Region graph

- 以 $s \equiv s'$ 的等價關係，將狀態空間分割成有限數目的子空間。
- $[s]$：所有與 s 等價的狀態，所構成的子狀態空間。
  - 每個等價的子空間，稱為一個region。
- $[s]$、 $[s']$之間的transition，
  - 可以代表自動機上的狀態轉換，
  - 也可以代表時間的前進。

TCTL模型檢驗問題

可以證明，以region graph就可以充分必要的回答TCTL的模型檢驗問題。

- 但須加上一個附加的clock變數 **z**，
  - $A,(q,\nu) \models \exists\, \varphi_1 U_{\sim c}\, \varphi_2$ iff $A,(q,\nu[z]) \models \exists\, \varphi_1 U_{\sim c}\, \varphi_2$
  - $\nu[z]$與ν完全一樣，除了$\nu[z](z)= 0$
- $A,(q,\nu[z]) \models \exists\, \varphi_1 U_{\sim c}\, \varphi_2$ iff存在由$(q,\nu[z])$開始的region序列 $r_0 r_1 r_2 \ldots r_k$
  - $r_0 r_1 r_2 \ldots r_{k-1}$ 都滿足 φ_1
  - r_k滿足 φ_2與 $z \sim c$ 。

TCTL的驗證問題複雜度

- TCTL與真時自動機的模型檢驗問題是PSPACE-complete。
- TCTL的satisfiability問題，是undecidable。

Clock Zones

- Finite representation of infinite state-space
- Conjunction of inequalities such as:

$$x < \mathrm{c} \quad | \quad x \leq \mathrm{c} \quad | \quad x - y < \mathrm{c} \quad | \quad x - y \leq \mathrm{c}$$

$$x, y \in X, \mathrm{c} \in \mathbf{Z}$$

- General form of a clock zone:

$$x_0 = 0 \wedge \bigwedge_{0 \leq i \neq j \leq \mathrm{n}} (x_i - x_j \sim \mathrm{c}_{ij})$$

$$\sim \in \{<, \leq\}$$

Clock Zones (Operations)

- Clock zones are closed under 3 operations:
- Let z_1, z_2 be two clock zones, $Y \subseteq X$, $t \geq 0$
- **Intersection**: $\boldsymbol{z_1 \wedge z_2}$ is a clock zone
  (Conjunction of conjunctions)
- **Clock Reset**: $\boldsymbol{z_1(Y{:=}0)}$ is a clock zone
  (Proof on pages 284 ~ 287)
- **Time Elapse**: $\boldsymbol{z_1 + t}$ is a clock zone
  (See next page)

Clock Zones (Time Elapse)

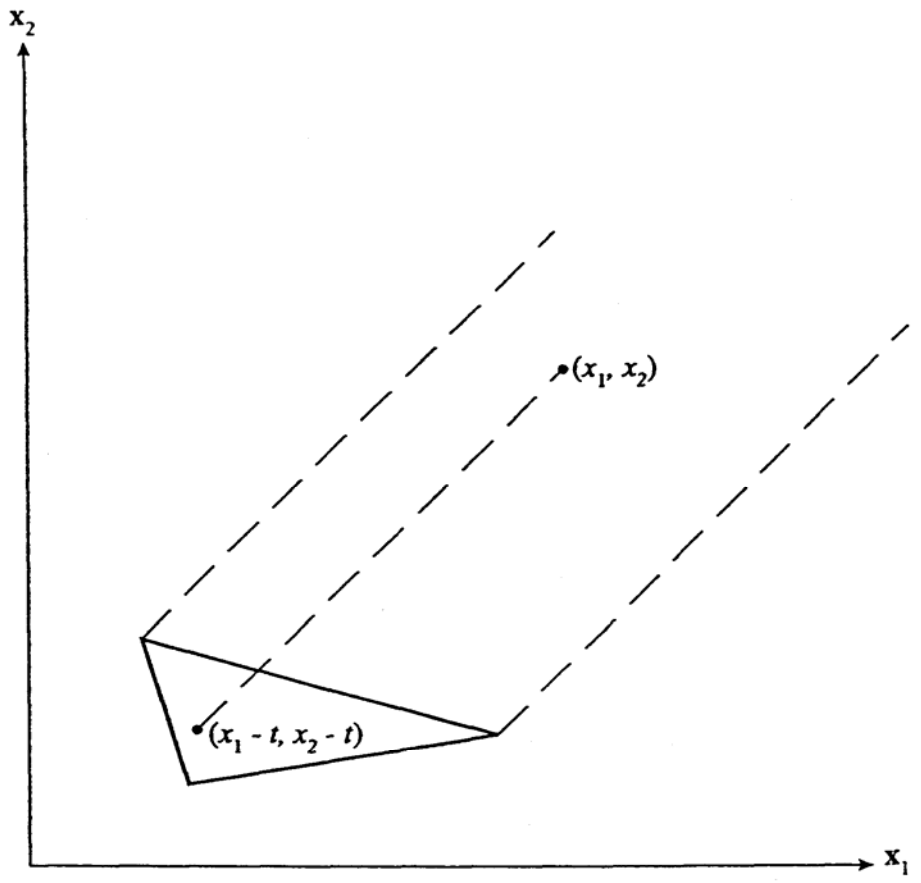


Figure 17.8
The clock zones φ and $\varphi^{\Uparrow}$.

Zones = Symbolic States

- **Zone**: **(*s*, *z*)**, where *s*: location, *z*: clock zone represents a symbolic state.
- **Successor Clock Zone**: **$z' = succ(z, e)$**, where *z*: clock zone, *e*: transition

  (successor clock zone obtained from *z* after time elapse and executing transition *e*)
- **Successor Zone**: **$(s', succ(z, e))$**, where *e*: $s \rightarrow s'$

Clock Zone (succ(z, e))

To obtain Successor Clock Zone (succ(z, e))

- Intersect z with $\mu(s)$
- Let time elapse in s (operator te())
- Intersect with $\mu(s)$
- Intersect with $\tau(e)$
- Reset all clocks from $\pi(e)$

$$\text{succ}(z, e) = ((\text{te}(z \wedge \mu(s)) \wedge \mu(s) \wedge \tau(e))[\pi := 0])$$

Closed under $\wedge$, te(), reset, also a **clock zone**!!!

Zone Graph

- Zone Graph is a transition system $Z(A)$
- States = zones of A
- Initial state = (s, [X := 0])
- For each transition e of A:

  a transition: $(s, z) \rightarrow (s', succ(z, e))$
- Zone reachability ➔State reachability

Difference Bound Matrix (DBM)

- DBM is a matrix to represent a clock zone

	0	x_1	$\ldots$	x_n
0		$< c$		
x_1	$< \infty$			$\leq r$
$\ldots$				
x_n	$\leq d$			

$0 - x_1 < c$,
i.e., $x_1 > c$

$x_1 < \infty$

$x_1 - x_n \leq r$

$x_n \leq d$

DBM (Example)

- $x_1 - x_2 < 2 \;\wedge\; 0 < x_2 \leq 2 \;\wedge\; 1 \leq x_1$

	0	x_1	x_2
0	≤ 0	≤ -1	< 0
x_1	$< \infty$	≤ 0	< 2
x_2	≤ 2	$< \infty$	≤ 0

DBM (Uniqueness)

- A zone is not uniquely represented by a DBM
- Zone: $x_1 - x_2 < 2 \;\wedge\; 0 < x_2 \leq 2 \;\wedge\; 1 \leq x_1$
- $x_1 - x_2 < 2$ and $x_2 \leq 2$ ➔ $x_1 < 4$

	0	x_1	x_2
0	≤ 0	≤ -1	< 0
x_1	$\boxed{< \infty}$	≤ 0	< 2
x_2	≤ 2	$< \infty$	≤ 0

	0	x_1	x_2
0	≤ 0	≤ -1	< 0
x_1	$\boxed{< 4}$	≤ 0	< 2
x_2	≤ 2	$< \infty$	≤ 0

DBM (Canonical Form)

- Canonical (unique) form of DBM for a zone
- Tightening operation:

$$\boxed{x_i - x_j \sim_{ij} d_{ij}} \text{ and } \boxed{x_j - x_k \sim_{jk} d_{jk}} \Rightarrow \boxed{x_i - x_k \sim_{ik} d_{ik}}$$

where $d_{ik} = d_{ij} + d_{jk}$

$\sim_{ik} = \leq$ if both $\sim_{ij}$ and $\sim_{jk}$ are $\leq$

$<$ otherwise

- Apply tightening operations to a DBM until no more change is possible!

DBM (Emptiness)

- Check if all elements on main diagonal are (≤ 0)
  - Yes ➔ nonempty
  - No ➔ empty or unsatisfiable
- Empty or unsatisfiable ➔
  At least 1 negative entry on main diagonal
- E.g. $\boxed{x_i - x_i \leq -1}$ ➔ $\boxed{0 \leq -1}$ ➔ FALSE!!!

DBM (3 operations: Intersection)

- Intersection: $D = D_1 \wedge D_2$ (all DBMs)
- Let $D_1(i, j) = \sim_1 c_1$ and $D_2(i, j) = \sim_2 c_2$

$$\boxed{D(i, j) = (\min(c_1, c_2), \sim)}$$

where
$$\begin{aligned} \sim &= \sim_1 \quad \text{if} \quad c_1 < c_2 \\ \sim &= \sim_2 \quad \text{if} \quad c_2 < c_1 \\ \sim &= \sim_1 \quad \text{if} \quad c_1 = c_2 \text{ and } \sim_1 = \sim_2 \\ \sim &= < \quad \text{if} \quad c_1 = c_2 \text{ and } \sim_1 \neq \sim_2 \end{aligned}$$

DBM (3 operations: Clock Reset)

- Clock Reset: $\boldsymbol{D' = D\,[Y := 0], \quad Y \subseteq X}$
  defined as follows:
  - $D'(i, j) = (\leq 0)$ if $x_i, x_j \in Y$
  - $D'(i, j) = D(0, j)$ if $x_i \in Y$ and $x_j \notin Y$
  - $D'(i, j) = D(i, 0)$ if $x_j \in Y$ and $x_i \notin Y$
  - $D'(i, j) = D(i, j)$ if $x_j \notin Y$ and $x_i \notin Y$

DBM (3 operations: Time Elapse)

- Time Elapse: $D' = \mathrm{te}(D)$ defined as follows:
- $D'(i, 0) = (< \infty)$, for any $i \neq 0$
- $D'(i, j) = D(i, j)$, for $i = 0$ or $j \neq 0$

	0	x_1	x_2
0	≤ 0	≤ -1	< 0
x_1	< 4	≤ 0	< 2
x_2	≤ 2	$< \infty$	≤ 0

	0	x_1	x_2
0	≤ 0	≤ -1	< 0
x_1	$< \infty$	≤ 0	< 2
x_2	$< \infty$	$< \infty$	≤ 0

DBM (3 operations)

- All 3 operations can be efficiently implemented
- DBM must be canonalized (using tightening) before any of the 3 operations (intersection, clock reset, and time elapse)
- After any of the 3 operations, a DBM might no longer be canonical!
- Last step: Reduce to canonical form!!!

DBM (Zone Graph Construction)

- Clock zones: represented by DBM
- Successor clock zones succ(z, e): computed by the 3 operations: intersection, reset, and time elapse on DBM instead of on clock zones

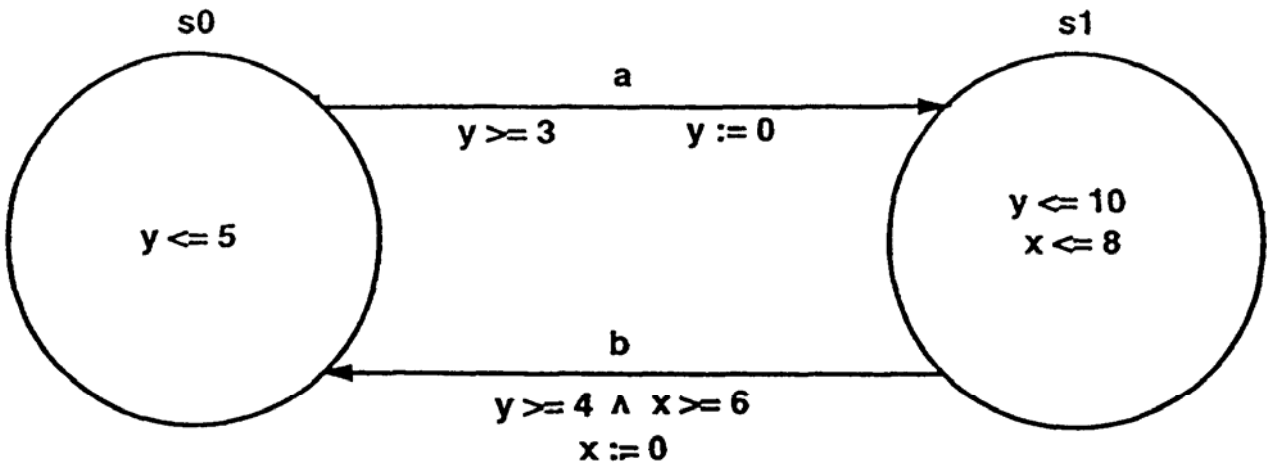


Figure 17.1
A simple timed automaton.

DBM (Zone Graph Construction)

- Initial state: (s0, Z0), Z0: $x = 0 \wedge y = 0$

	0	x	y
0	≤ 0	≤ 0	≤ 0
x	≤ 0	≤ 0	≤ 0
y	≤ 0	≤ 0	≤ 0

Zone D_0

DBM (Zone Graph Construction)

- Invariant μ (s0) is $0 \leq x \ \wedge \ 0 \leq y \leq 5$

	0	x	y
0	≤ 0	≤ 0	≤ 0
x	$< \infty$	≤ 0	$< \infty$
y	≤ 5	≤ 5	≤ 0

DBM (Zone Graph Construction)

- Step 1: Intersection D_0 with μ (s0) gives D_0
- Step 2: Let time elapse: te($D_0 \wedge \mu$ (s0))

	0	x	y
0	≤ 0	≤ 0	≤ 0
x	≤ 0	≤ 0	≤ 0
y	≤ 0	≤ 0	≤ 0

	0	x	y
0	≤ 0	≤ 0	≤ 0
x	$< \infty$	≤ 0	≤ 0
y	$< \infty$	≤ 0	≤ 0

DBM (Zone Graph Construction)

- Step 3: Intersect with μ (s0) again

	0	x	y
0	≤ 0	≤ 0	≤ 0
x	$< \infty$	≤ 0	≤ 0
y	$< \infty$	≤ 0	≤ 0

$\Rightarrow$

	0	x	y
0	≤ 0	≤ 0	≤ 0
x	≤ 5	≤ 0	≤ 0
y	≤ 5	≤ 0	≤ 0

DBM (Zone Graph Construction)

- Trigger $\tau(a) = y \geq 3$

	0	x	y
0	≤ 0	≤ 0	≤ -3
x	$< \infty$	≤ 0	$< \infty$
y	$< \infty$	$< \infty$	≤ 0

DBM (Zone Graph Construction)

- Step 4: Intersect with trigger τ(a)

	0	x	y
0	≤ 0	≤ 0	≤ 0
x	≤ 5	≤ 0	≤ 0
y	≤ 5	≤ 0	≤ 0

	0	x	y
0	≤ 0	≤ -3	≤ -3
x	≤ 5	≤ 0	≤ 0
y	≤ 5	≤ 0	≤ 0

DBM (Zone Graph Construction)

- Step 5: Reset clock y in DBM

	0	x	y
0	≤ 0	≤ -3	≤ -3
x	≤ 5	≤ 0	≤ 0
y	≤ 5	≤ 0	≤ 0

$\Rightarrow$

	0	x	y
0	≤ 0	≤ -3	≤ 0
x	≤ 5	≤ 0	≤ 5
y	≤ 0	≤ -3	≤ 0

$$Z1 = 3 \leq x \leq 5 \wedge 3 \leq x - y \leq 5 \wedge y = 0$$

DBM (Zone Graph Construction)

- Successor of (s0, Z0) is (s1, Z1)
- Repeat the same 5 steps:
- (s0, $4 \leq y \leq 5 \wedge 4 \leq y - x \leq 5 \wedge x = 0$)
- (s1, $0 \leq x \leq 1 \wedge 0 \leq x - y \leq 1 \wedge y = 0$)
- (s0, $5 \leq y \leq 8 \wedge 5 \leq y - x \leq 8 \wedge x = 0$)
- (s1, $x = 0 \wedge y = 0$),
  which is contained in the 2nd zone, thus no more new zones can be generated!!!
  No more change! Stop! Zone Graph!

Binary Decision Diagram (BDD)

- BDD: A canonical form representation for Boolean formulas.
- Motivation:
  - Too much space redundancy in traditional representations
  - BDD is more compact than truth tables, conjunctive normal form, disjunctive normal form, binary decision trees, etc.
  - BDD has a canonical form
  - BDD operations are efficient

BDD (Binary Decision Tree)

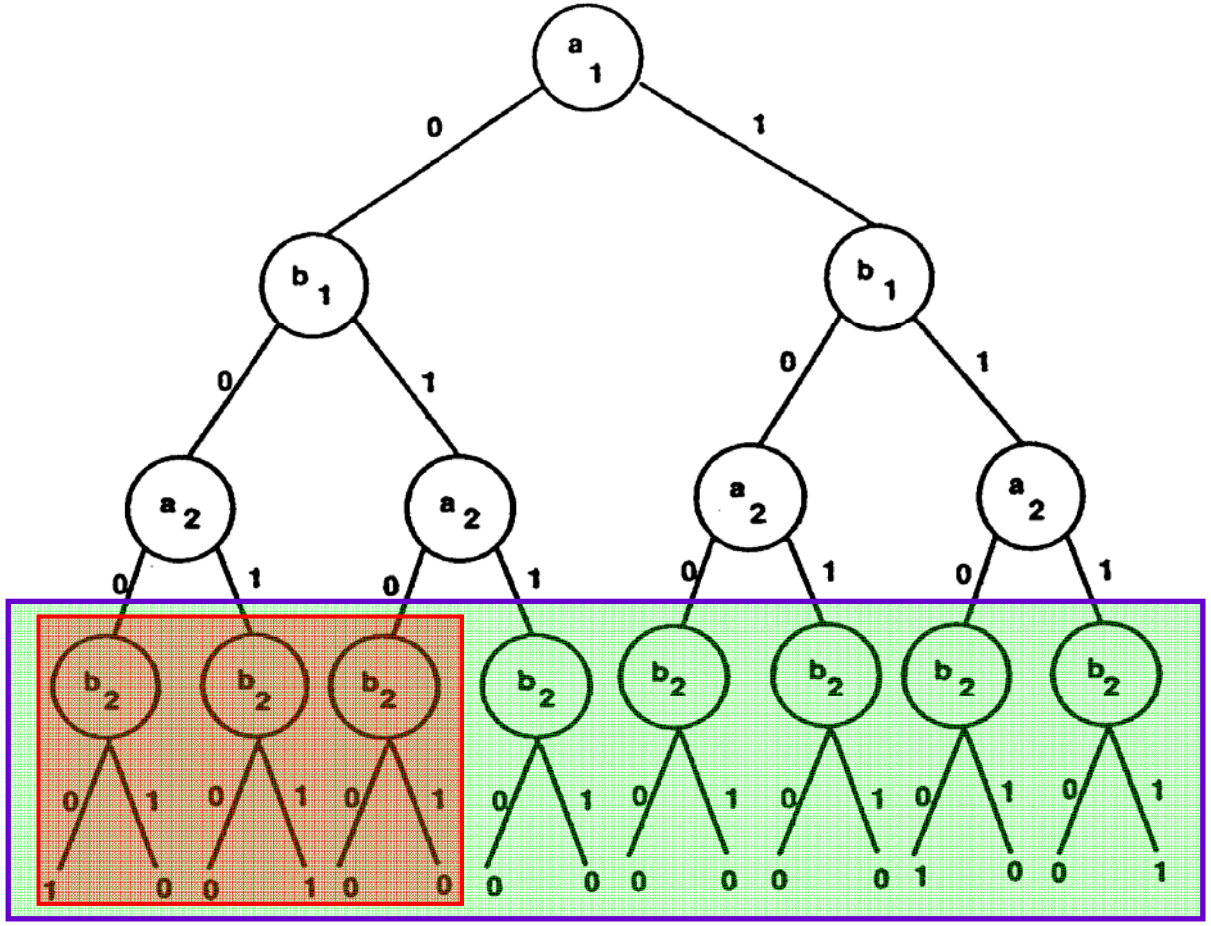


Figure 5.1
Binary decision tree for two-bit comparator.

BDD (redundancy in BDT)

- Binary Decision Trees (BDT):
  - Same size as truth tables
  - Lots of redundancy: Out of 8 subtrees rooted at b_2 only 3 are distinct!!!
  - Merge isomorphic subtrees ➔ BDD
- BDD is a rooted, DAG with 2 types of vertices: terminal and nonterminal
- Each nonterminal *v* is labeled with var(*v*) and has two successors: low(*v*) and high(*v*)
- Each terminal vertex is labeled 0 or 1

BDD (Boolean Function)

- BDD B with root v determines Boolean function $f_v(x_1, \ldots, x_n)$ as follows:
- If v is a terminal vertex

$$f_v(x_1, \ldots, x_n) = \text{value}(v) \in \{0, 1\}$$

- If v is a nonterminal vertex with $\text{var}(v) = x_i$:

$$f_v(x_1, \ldots, x_n) = (\neg x_i \wedge f_{low(v)}(x_1, \ldots, x_n)) \vee (x_i \wedge f_{high(v)}(x_1, \ldots, x_n))$$

BDD (Canonical Form)

- Canonical Form:

  Two boolean functions are logically equivalent iff they have isomorphic representations

- Why Canonical Form?

  Simplifies equivalence checking and satisfiability checking

- BDD Canonical Form:

  1. Same variable order along all paths from root to terminal
  2. No isomorphic subtrees or redundant vertices

BDD (Canonical Form)

- For requirement 1 (same variable order):
  - Total ordering < on all variables
  - If u has a nonterminal successor v, then $\mathrm{var}(u) < \mathrm{var}(v)$
- For requirement 2 (no redundancy): Apply **3 transformations repeatedly**:
  - Remove duplicate terminals
  - Remove duplicate non-terminals
  - Remove redundant tests

BDD (Canonical Form)

- Remove duplicate terminals: Eliminate all but one terminal vertex with a given label and redirect hanging arcs.
- Remove duplicate non-terminals: If var(u) = var(v), low(u) = low(v), high(u) = high(v), then eliminate u or v. Redirect hanging arcs.
- Remove redundant tests: If low(v) = high(v), then eliminate v and redirect incoming arcs to low(v).

BDD (Canonical Form)

- Apply 3 transformations repeatedly in a bottom-up manner
- Time: O(|BDD|)
- Gives Ordered BDD (OBDD)
- Order: $a_1 < b_1 < a_2 < b_2$
- OBDD for two-bit comparator example

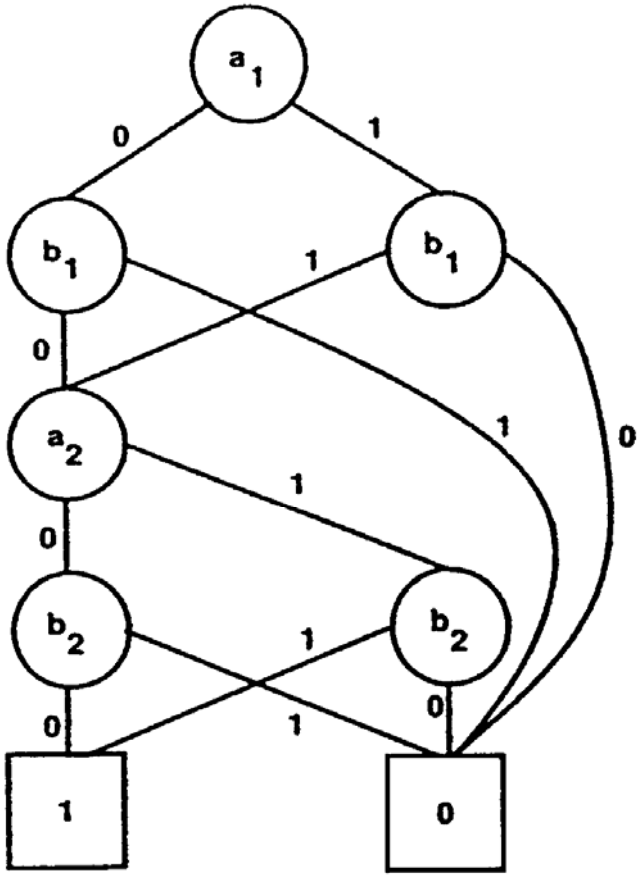


Figure 5.2
OBDD for two-bit comparator.

Ordered BDD (OBDD)

- Since OBDDs are canonical, it is easy to:
  - check equivalence = check BDD isomorphism
  - check satisfiability = check BDD isomorphism with OBDD(0)
- Size of OBDD depends critically on ***VARIABLE ORDERING***!!!
- 2-bit comparator example:
  Change variable order to: $a_1 < a_2 < b_1 < b_2$
  11 vertices instead of 8 for $a_1 < b_1 < a_2 < b_2$

OBDD (Variable Ordering)

- $a_1 < a_2 < b_1 < b_2$
- In general, for n-bit comparator:

  $a_1 < b_1 < \ldots < a_n < b_n$
  gives $3n + 2$ vertices

  $a_1 < \ldots < a_n < b_1 < \ldots < b_n$
  gives $3 \times 2^n - 1$ vertices

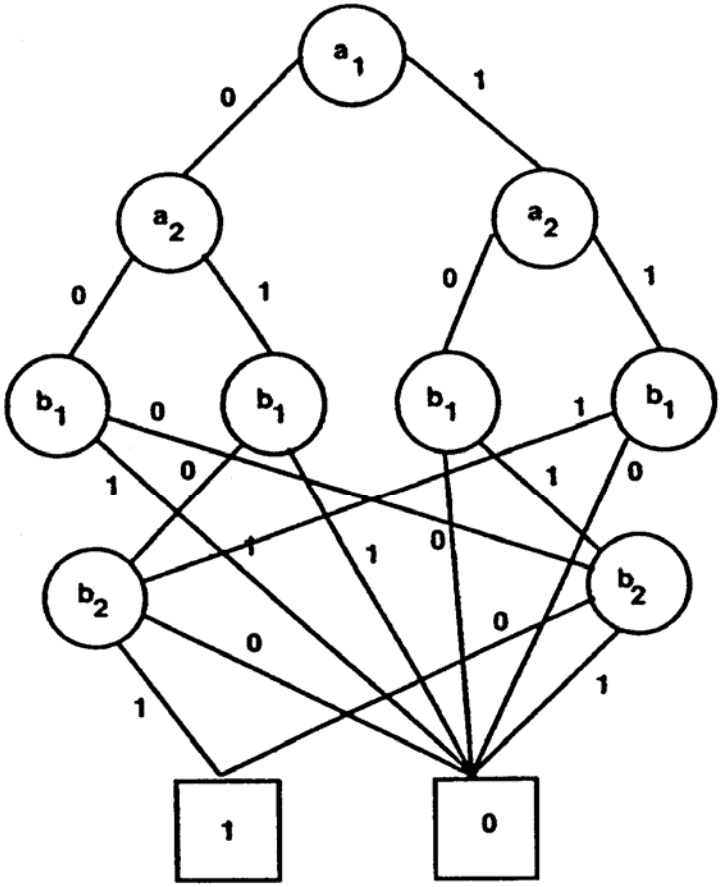


Figure 5.3
OBDD for two-bit comparator.

OBDD (Variable Ordering)

- Infeasible to find an optimal variable ordering!
- Checking a particular ordering is optimal is NP-complete!
- There are Boolean functions with exponential size OBBDs for any ordering!
- *Eg*: n^{th} output of a combinational circuit to multiply two n bit integers

OBDD (Variable Ordering)

- **Heuristics** to find a good variable ordering:
- *Depth-First Traversal* of circuit gives a good variable order
- Intuition: related variables should be grouped together in the order
- Dynamic Reordering:
  - to save time and space,
  - not to find an optimal ordering.

OBDD (Operations)

- All 16 two-argument logical operations can be implemented efficiently on OBDDs.
- Time Complexity = $O(|OBDD_1| \times |OBDD_2|)$
- Key Idea: Shannon Expansion

$$f = (\neg x \wedge f|_{x:=0}) \vee (x \wedge f|_{x:=1})$$

- Let f, f' = OBDDs, v, v' = roots,
  $x = \mathrm{var}(v)$, $x' = \mathrm{var}(v')$
- Apply(f, f', *,): a uniform algorithm for all 16 operations *

OBDD (Apply Algorithm)

- If v, v' are terminals,

$$f * f' = \text{value}(v) * \text{value}(v')$$

- If $x = x'$, then use Shannon Expansion:

$$f * f' = (\neg x \wedge (f|_{x:=0} * f'|_{x:=0})) \vee (x \wedge f|_{x:=1} * f'|_{x:=1}))$$

- If $x < x'$, then Shannon Expansion simplifies:

$$f * f' = (\neg x \wedge (f|_{x:=0} * f')) \vee (x \wedge f|_{x:=1} * f'))$$

$(\because f'|_{x:=0} = f'|_{x:=1} = f'$ does not depend on $x)$

- If $x' < x$, same as previous.

OBDD (Efficiency)

- A problem generates 2 sub-problems
- Use dynamic programming to avoid exponential algorithm
- # sub-graphs = $|\mathrm{OBDD}(f)|$
- A sub-problem depends on 2 sub-graphs
- # sub-problems $\leq |\mathrm{OBDD}(f_1)| \times |\mathrm{OBDD}(f_2)|$
- Result Cache: computed sub-problems in canonical form
- Modern BDD package: millions of vertices!

OBDD (Representing Kripke Structures)

- State variables: a, b
- Add new state variables: a', b'
- $s_1 \rightarrow s_2$: $(a \wedge b \wedge a' \wedge \neg b')$
- Full system:
  $(a \wedge b \wedge a' \wedge \neg b') \vee (a \wedge \neg b \wedge a' \wedge \neg b') \vee$
  $(a \wedge \neg b \wedge a' \wedge b')$

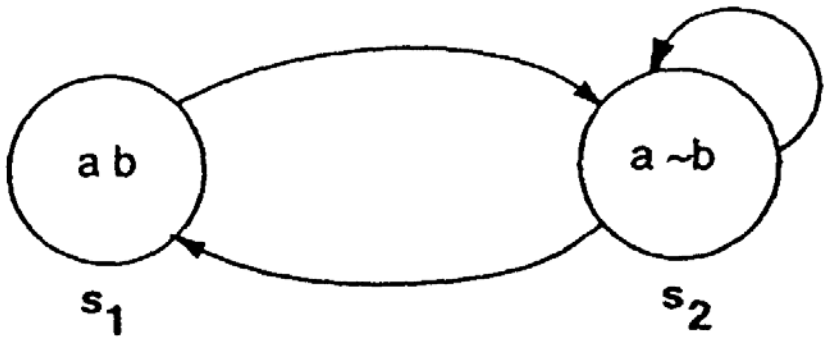
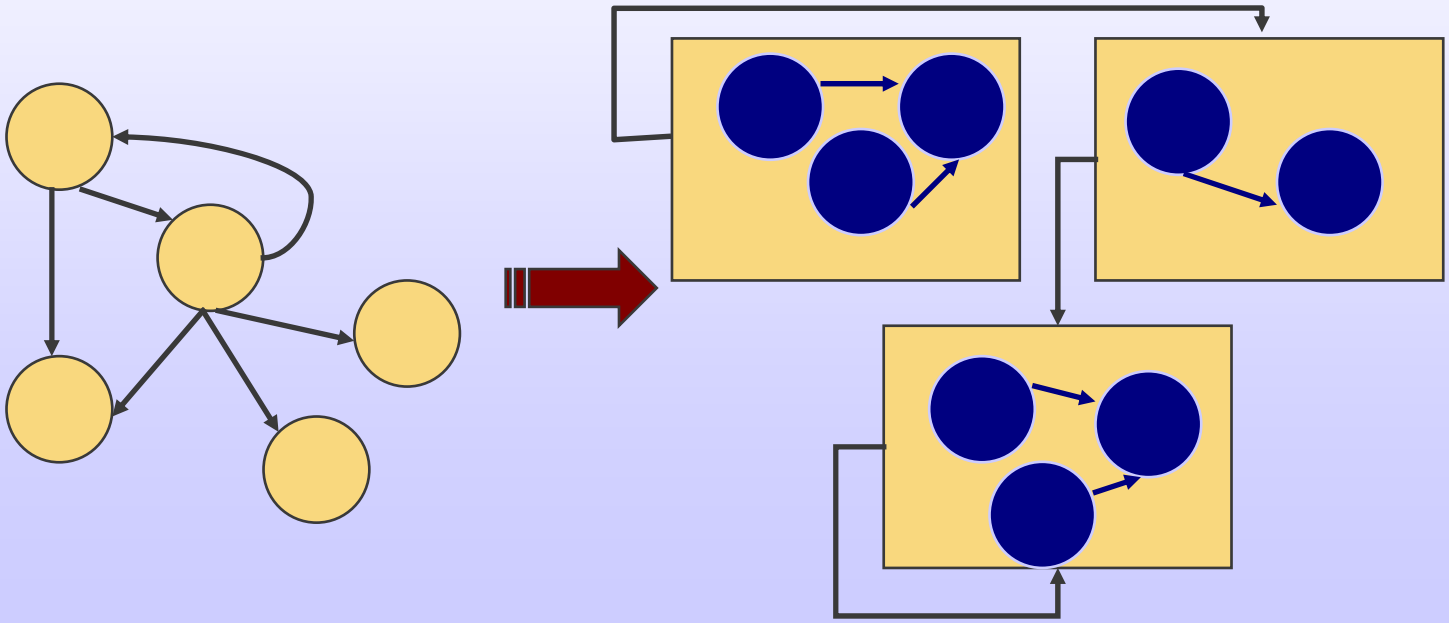


Figure 5.4
Two-state Kripke structure.

Symbolic Model Checking

- Symbolic = Manipulation of Boolean formulas
- Symbolic ➔ Operate on entire sets of states instead of individual states and transitions



Fixpoint Representations

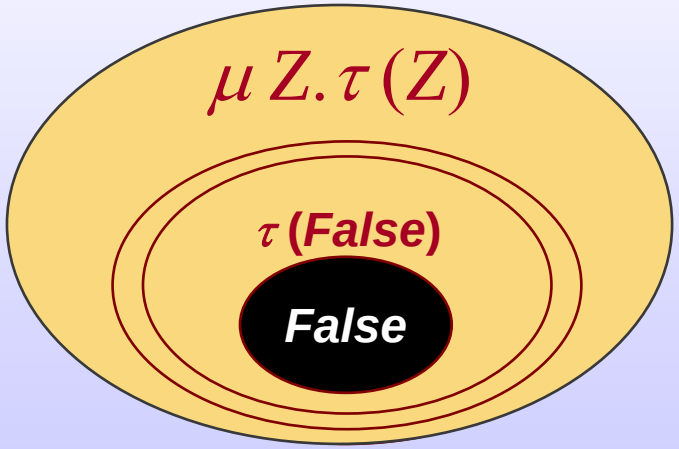
- $M(S, R, L)$: Finite Kripke Structure
- $P(S)$: power set of S
- Least element in $P(S)$: $False$ = empty set
- Greatest element in $P(S)$: $True = S$
- Predicate transformer: $\tau : P(S) \rightarrow P(S)$
- $\tau^0(Z) = Z,\ \tau^{i+1}(Z) = \tau^i(Z)$
- τ is monotonic if $P \subseteq Q \Rightarrow \tau(P) \subseteq \tau(Q)$

Fixpoint Representations

- Fixpoint characterization of temporal logic operators
- A set $S' \subseteq S$ is a fixpoint of a function $\tau : P(S) \rightarrow P(S)$ if $\tau(S') = S'$.
  (S: set of states, $P(S)$: power set of S)
- A monotonic predicate transformer τ on $P(S)$ always has:
  - a least fixpoint: $\mu Z.\tau(Z)$
  - a greatest fixpoint: $\nu Z.\tau(Z)$

Least Fixpoint

- $False \subseteq \tau(False) \subseteq \tau^2(False) \subseteq \ldots$
- No. of iterations $\leq |S|$
- Last iteration: $\tau(Q) = Q = \mu Z.\tau(Z)$



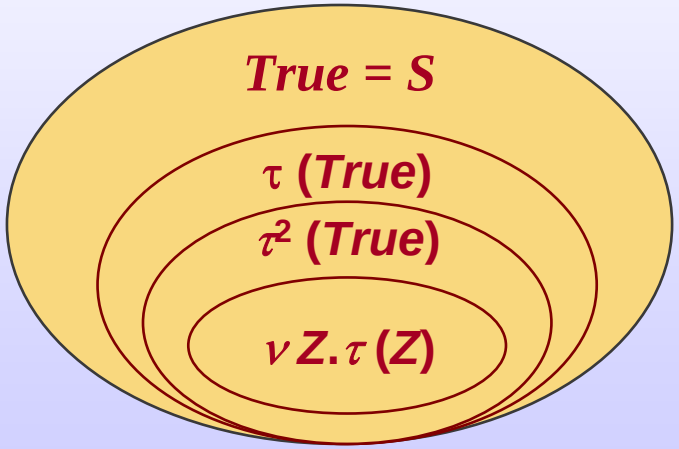
Least Fixpoint Algorithm

```
function Lfp(Tau : PredicateTransformer) : Predicate
        Q := False;
        Q' := Tau(Q);
        while (Q ≠ Q') do
                Q := Q';
                Q' := Tau(Q');
        end while;
        return(Q);
end function
```

Figure 6.1
Procedure for computing least fixpoints.

Greatest Fixpoint

- $True \supseteq \tau(True) \supseteq \tau^2(True) \supseteq \ldots$
- No. of iterations $\leq |S|$
- Last iteration: $\tau(Q) = Q = \nu Z.\tau(Z)$



Greatest Fixpoint Algorithm

```
function Gfp(Tau : PredicateTransformer) : Predicate
        Q := True;
        Q' := Tau(Q);
        while (Q ≠ Q') do
                Q := Q';
                Q' := Tau(Q');
        end while;
        return(Q);
end function
```

Figure 6.2
Procedure for computing greatest fixpoints.

Fixpoint Characterizations

- $\forall\Diamond\varphi = \mu Z\,.\,\varphi \vee \forall\bigcirc Z$
- $\exists\Diamond\varphi = \mu Z\,.\,\varphi \vee \exists\bigcirc Z$
- $\forall\Box\varphi = \nu Z\,.\,\varphi \wedge \forall\bigcirc Z$
- $\exists\Box\varphi = \nu Z\,.\,\varphi \wedge \exists\bigcirc Z$
- $\forall[\varphi_1\ U\ \varphi_2] = \mu Z\,.\,\varphi_2 \vee (\varphi_1 \wedge \forall\bigcirc Z)$
- $\exists[\varphi_1\ U\ \varphi_2] = \mu Z\,.\,\varphi_2 \vee (\varphi_1 \wedge \exists\bigcirc Z)$

$\Diamond \rightarrow \mu Z \qquad \Box \rightarrow \nu Z$

Approximations for $\exists[p\ U\ q]$

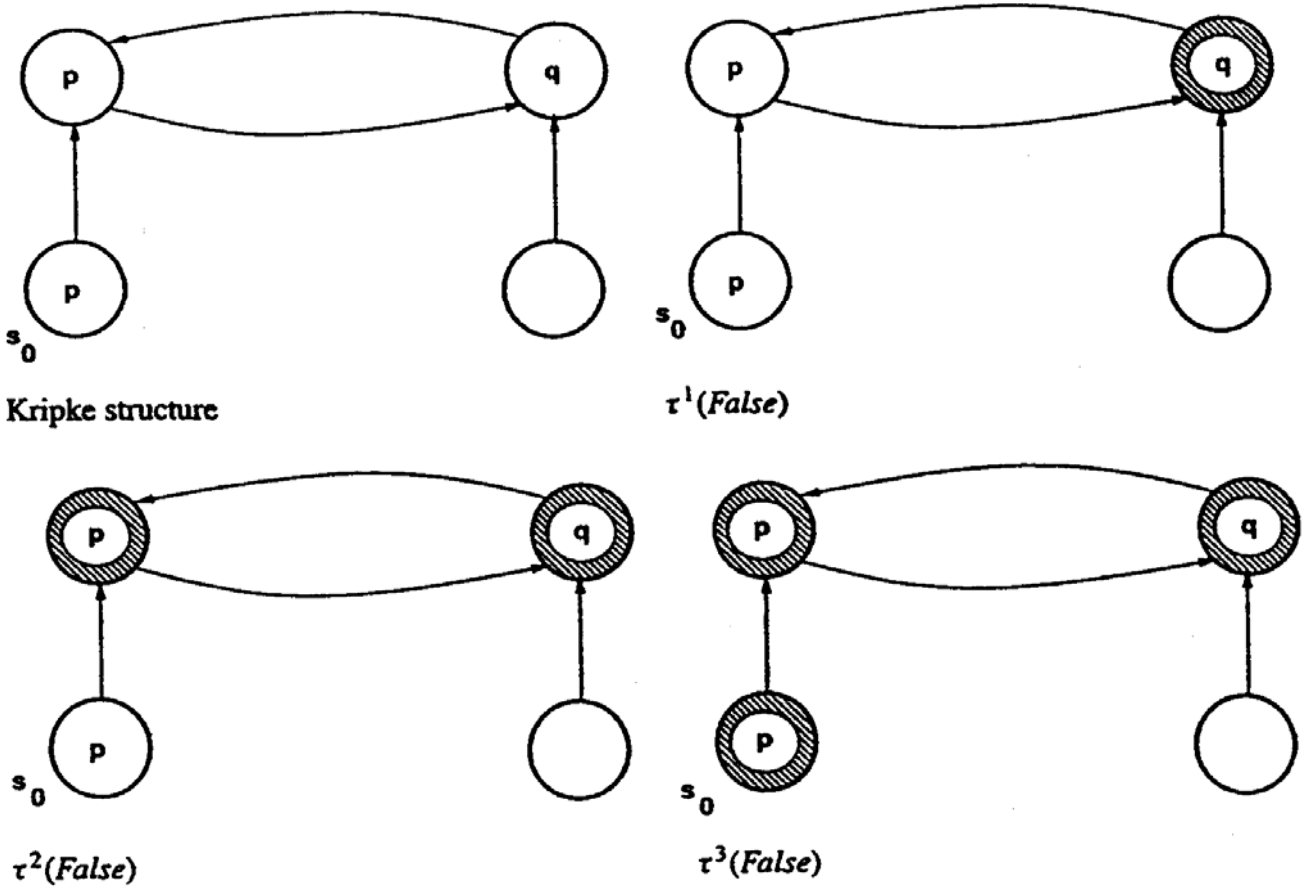


Figure 6.3
Sequence of approximations for $\mathbf{E}[p\ \mathbf{U}\ q]$.

Symbolic Model Checking for CTL

- Kripke Structures are represented symbolically using OBDDs.
- Quantified Boolean Formulas QBF(V):
  - every propositional variable in V is a formula
  - $\neg f$, $f \vee g$, $f \wedge g$ are formulas, if f and g are
  - $\exists v\, f$ and $\forall v\, f$ are formulas, if f is and $v \in V$
- Every QBF formula can be represented by an OBDD

$$\exists x\, f = f|_{x=0} \vee f|_{x=1}$$

$$\forall x\, f = f|_{x=0} \wedge f|_{x=1}$$

Symbolic Model Checking Algorithm

- Check(a CTL formula) returns an OBDD that represents all system states satisfying the formula
- Inductive definition of Check() as follows:
  - If $f = a$ (a proposition), Check(f) returns the set of states satisfying a.
  - If $f = f_1 \wedge f_2$ or $f = \neg f_1$, use Apply(Check(f_1), Check(f_2), *)

Symbolic Model Checking Algorithm

- Check($\exists \bigcirc f$) = CheckEX(Check(f))
- Check($\exists [f\ U\ g]$) =
  CheckEU(Check(f), Check(g))
- Check($\exists \Box f$) = CheckEG(Check(f))

CTL formula

OBDD

CheckEX(f)

- $\exists \bigcirc f$ is true in a state if it has a successor in which f is true.
- $\text{CheckEX}(f(v)) = \exists v'[f(v') \wedge R(v, v')]$
  $= [f(v') \wedge R(v, v')]|_{v'=0} \vee [f(v') \wedge R(v, v')]|_{v'=1}$
- R is the transition relation OBDD
- Given OBDDs for f and R, we can compute an OBDD for $\exists v'[f(v') \wedge R(v, v')]$

CheckEU(f)

- Least fixpoint characterization:
  $\exists[\varphi_1 \; U \; \varphi_2] = \mu Z \,.\, \varphi_2 \vee (\varphi_1 \wedge \exists \bigcirc Z)$
- Use Lfp() to compute $Q_0 = False$, Q_1, …, Q_i, … that converges to $\exists[\varphi_1 \; U \; \varphi_2]$
- Given OBDDs for φ_1, φ_2, and Q_i, we can compute Q_{i+1}.
- $Q_{i+1} = \varphi_2 \vee (\varphi_1 \wedge \exists \bigcirc Q_i)$
- $Q_{i+1} = Q_i \;\; \rightarrow \;\; Q_i = \mu Z \,.\, \varphi_2 \vee (\varphi_1 \wedge \exists \bigcirc Z)$

CheckEG(f)

- Greatest fixpoint characterization
  $\exists\Box\varphi = \nu Z\,.\,\varphi \wedge \exists\bigcirc Z$
- Use Gfp() to compute $Q_0 = True$, Q_1, …, Q_i, … that converges to $\exists\Box\varphi$
- Given OBDDs for φ and Q_i, we can compute Q_{i+1}.
- $Q_{i+1} = \varphi \wedge \exists\bigcirc\, Q_i$
- $Q_{i+1} = Q_i \;\rightarrow\; Q_i = \nu Z\,.\,\varphi \wedge \exists\bigcirc Z$

Fairness in Symbolic Model Checking

- Fairness Constraints $F = \{P_1, \ldots, P_n\}$
- $\exists\Box\varphi = \nu Z \,.\, \varphi \wedge \bigwedge_{k=1\ldots n} \exists\bigcirc\exists[\varphi\, U(Z \wedge P_k)]$
- Set of all states which are start of some fair computation is $fair(v) = \text{CheckFair}(\exists\Box True)$
- $\text{CheckFairEX}(f(v)) =$
  $\text{CheckEX}(f(v) \wedge fair(v))$
- $\text{CheckFairEU}(f(v), g(v)) =$
  $\text{CheckEU}(f(v), g(v) \wedge fair(v))$

Counterexamples & Witnesses

- Counterexample: If $\forall\Box\varphi$ is false, produce a path to a state in which $\neg\varphi$ holds.
- Witness: If $\exists\Diamond\varphi$ is true, produce a path to a state in which φ holds.
- Counterexample($\forall\Box\varphi$) = Witness($\exists\Diamond\neg\varphi$)
- Thus, need consider only witnesses:

  $\exists\Box$, $\exists U$, and $\exists\bigcirc$.

Witnesses

- Construct SCC graph G(V, E), where each node in V is an SCC, and an edge exists if there is a transition from a state in one SCC to a state in another SCC.
- SCC graph contains no cycle (because all cycles are in SCC graph nodes)
- Each infinite path has a suffix in an SCC graph node.

Witness for $\exists\Box\varphi$ with $F = \{P_1,\ldots,P_k\}$

- $\exists\Box\varphi = \nu Z\,.\,\varphi \wedge \bigwedge_{k=1\ldots n} \exists\bigcirc\exists[\varphi\, U(Z \wedge P_k)]$
- For each P, compute an increasing sequence of approximations $Q_0^P \subseteq Q_1^P \subseteq Q_2^P \subseteq \ldots$, where Q_i^P is the set of states from which a state in $Z \wedge P$ can be reached in $\leq i$ steps.
- Start with an initial state s satisfying $\exists\Box\varphi$.
- Look for a successor t of s in Q_0^P, $\forall P \in F$.
- If no such t, keep looking in Q_1^P, Q_2^P, …
- If $t \in Q_i^P$ and $i > 0$, find a successor of t in Q_{i-1}^P, and continue until $i = 0$, thus we have a path from s to some state u in $\exists\Box\varphi \wedge P$. Continue with other $P \in F$ to obtain s'. Find a path from s' to t (cycle).

Witness for $\exists\Box\varphi$ in first SCC

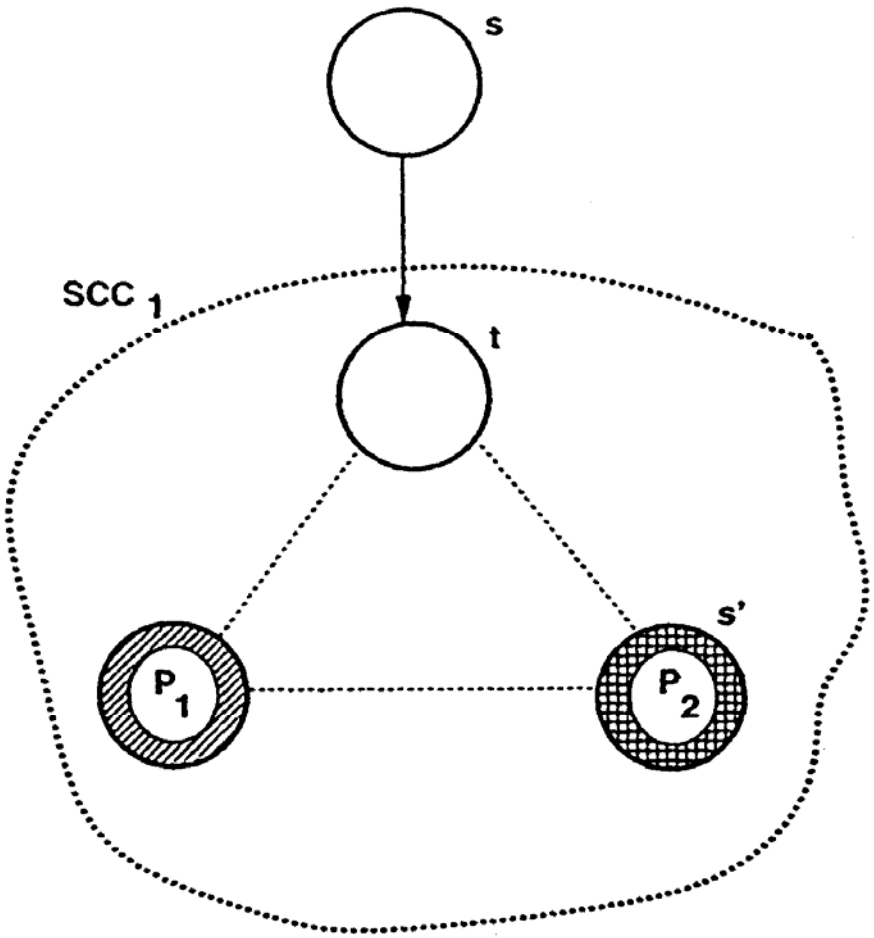


Figure 6.4
Witness is in the first strongly connected component.

Witness for $\exists\Box\varphi$ spans 3 SCC

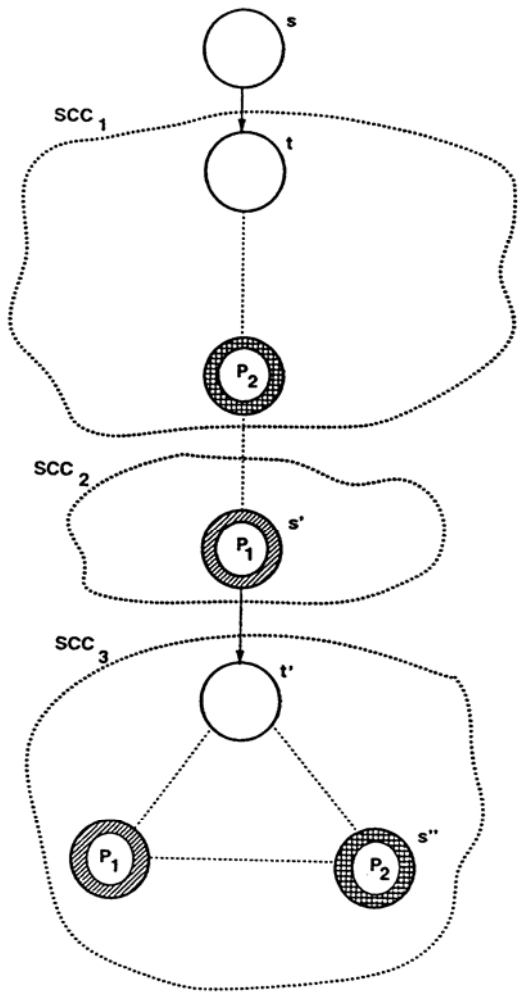


Figure 6.5
Witness spans three strongly connected components.

Witnesses for ∃U and ∃○

- *fair* = {s | state s satisfies ∃□*True* with F}
- Witness for ∃ [f U g] with F =
  Witness for ∃ [f U (g ∧ *fair*)]
- Witness for ∃○[f] with F =
  Witness for ∃○[f ∧ *fair*]

An ALU Example

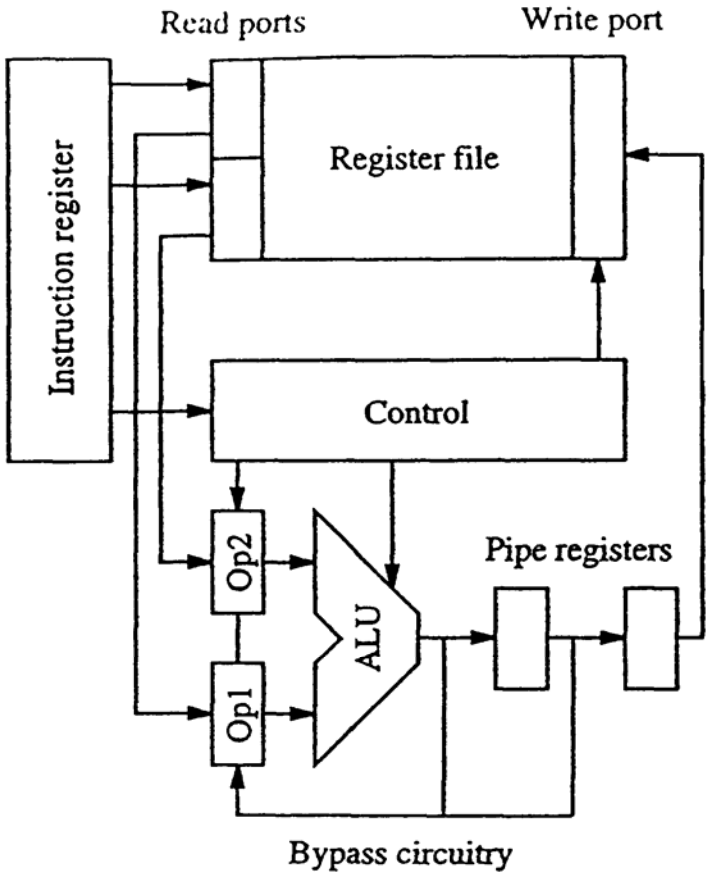


Figure 6.6
Pipeline circuit block diagram.

An ALU Example

Pipeline circuit

- First cycle of instruction: read operands from register file into operand registers
- Second cycle: compute result and store in first pipe register
- Third to $r + 1$ cycle: result is passed through remaining pipeline registers
- Last cycle: result is written to register file

An ALU Example

- Only after $r + 2$ cycles, the result of an operation is written back to register file.
- Input signal *stall* ➔ invalid instruction (e.g., cache miss) ➔ no change to destination reg
- Assume: 1 pipe register, XOR op only
- CTL specification has 2 parts:
  - Destination register updated correctly
  - All other registers should not change

An ALU Example

Spec Part 1 (Dest reg updated correctly)

- $\forall\Box(\neg stall \rightarrow ((src1op_i \oplus src2op_i) \equiv result_i))$
- Use $\forall\bigcirc^n$ operator for n cycle delays
- $src1op_i = (\neg src1addr \wedge \forall\bigcirc^2 reg_{0,i}) \vee (src1addr \wedge \forall\bigcirc^2 reg_{1,i})$
- $src2op_i = (\neg src2addr \wedge \forall\bigcirc^2 reg_{0,i}) \vee (src2addr \wedge \forall\bigcirc^2 reg_{1,i})$
- $result_i = (\neg destaddr \wedge \forall\bigcirc^3 reg_{0,i}) \vee (destaddr \wedge \forall\bigcirc^3 reg_{1,i})$

An ALU Example

Spec Part 2 (No change in other registers)

- $\forall\Box((stall \vee \neg destaddr) \rightarrow (\forall\bigcirc^2 reg_{1,i} \equiv \forall\bigcirc^3 reg_{1,i}))$
- $\forall\Box((stall \vee destaddr) \rightarrow (\forall\bigcirc^2 reg_{0,i} \equiv \forall\bigcirc^3 reg_{0,i}))$

An ALU Example (Experiment Results)

- 8-bit wide pipeline
- 4 general registers
- 1 pipeline register
- 1 operation
- More than 10^{20} states
- 41,000 OBDD nodes
- Verification time: 22 mins of Sun 3
- Substantial improvement over explicit-state model checking