



第十一章 基于形式化方法的服务质量保障

王旭

北京航空航天大学



软件形式化方法

- 采用数学上的形式化方法对软件进行抽象和描述A
 - finite state machines, labelled transition systems, Petri nets, vector addition systems, timed automata, hybrid automata, process algebra等
- 目的是对某些属性进行验证和证明
 - 属性a: 使用形式化方法进行描述
 - 推理或证明: $A \rightarrow a$? If not, counterexample



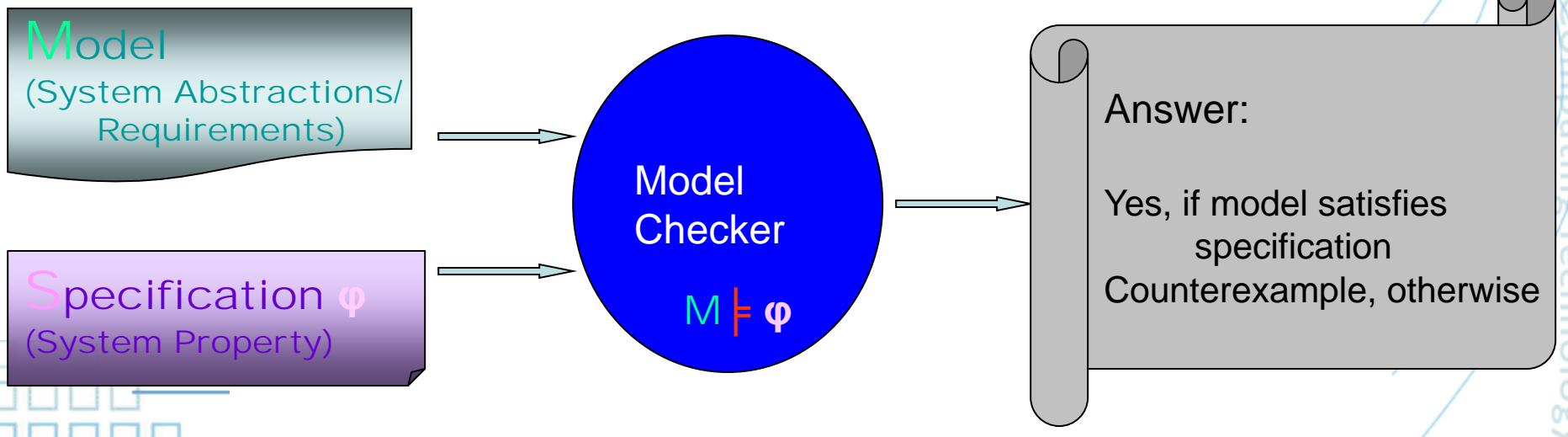
形式化验证方法

- 一般来说，不同的软件形式化方法、不同的属性形式化方法对应不同的形式化验证方法
- 两种常见形式化验证方法
 - 模型检测(Model Checker)
 - 软件描述为状态和状态的变化，通过对所有状态的枚举或搜索来判断某些属性是否满足
 - 演绎推理(Deductive Verification)
 - 将软件实现抽象为一系列逻辑语句，将属性描述为定理逻辑语句，最后借用定理证明器(theorem provers)或SMT求解器等进行演绎推理，从而判断定理是否为真



模型检测

- **模型(Model)**
 - 软件系统的抽象模型，一般是有限状态机
- **属性(Property)**
 - 形式化的规约(Specification)



属性

- **安全属性 Safety Properties**
 - Invariants, deadlocks, reachability, etc.
 - Invariants: “x is always less than 10”
 - Deadlock freedom: “the system never reaches a state where no moves are possible”
 - **“something bad never happens”**
- **活性属性 Liveness Properties**
 - Termination, response, etc.
 - Termination: “the system eventually terminates”
 - Response properties: “if action X occurs then eventually action Y will occur”
 - **“something good will eventually happen”**



信号互斥例子(Mutual Exclusion)

Model
(System Requirements)

The Model (Willem Visser, <http://ase.arc.nasa.gov/visser/ASE2002TutSoftwareMC-fonts.ppt>)

- Two process mutual exclusion with shared semaphore
- Each process has three states
 - Non-critical (N)
 - Trying (T)
 - Critical (C)
- Semaphore can be available (S_0) or taken (S_1)
- Initially both processes are in the Non-critical state and the semaphore is available --- $N_1 N_2 S_0$

$$\begin{array}{l} N_1 \rightarrow T_1 \\ T_1 \wedge S_0 \rightarrow C_1 \wedge S_1 \\ C_1 \rightarrow N_1 \wedge S_0 \end{array} \quad || \quad \begin{array}{l} N_2 \rightarrow T_2 \\ T_2 \wedge S_0 \rightarrow C_2 \wedge S_1 \\ C_2 \rightarrow N_2 \wedge S_0 \end{array}$$

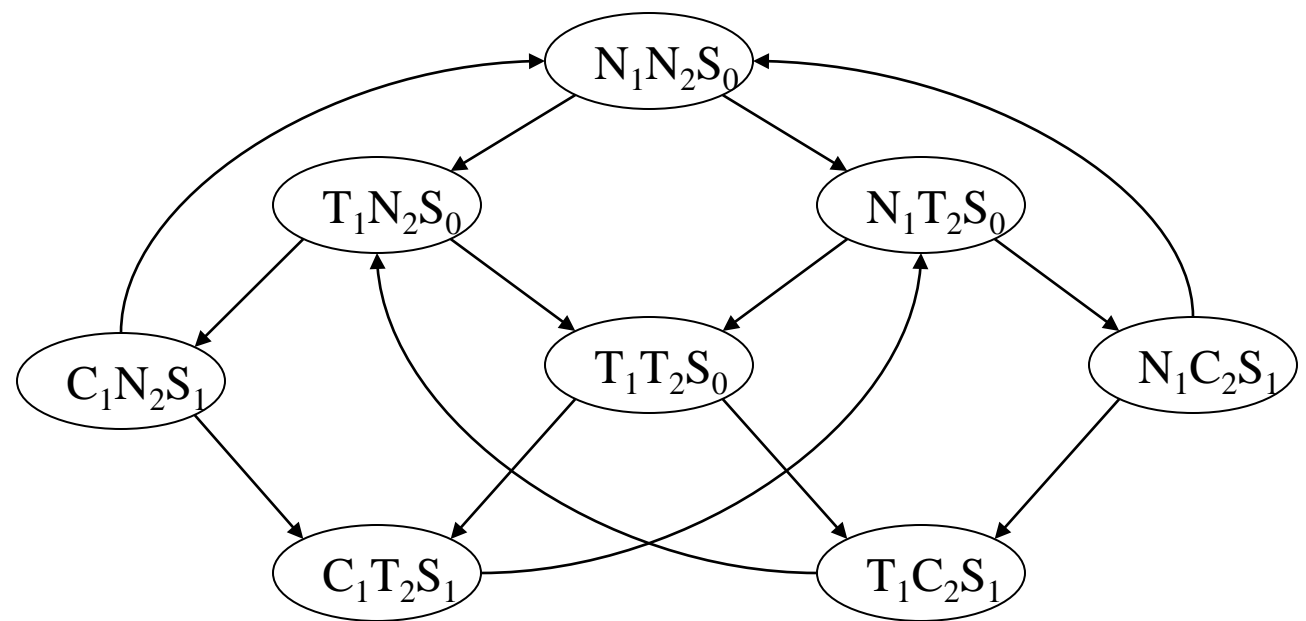
信号互斥例子(Mutual Exclusion)

Model
(System Requirements)

The Model (Willem Visser, <http://ase.arc.nasa.gov/visser/ASE2002TutSoftwareMC-fonts.ppt>)

- Initially both processes are in the Non-critical state and the semaphore is available --- $N_1 N_2 S_0$

$$\begin{array}{l} N_1 \rightarrow T_1 \\ T_1 \wedge S_0 \rightarrow C_1 \wedge S_1 \\ C_1 \rightarrow N_1 \wedge S_0 \end{array} \quad || \quad \begin{array}{l} N_2 \rightarrow T_2 \\ T_2 \wedge S_0 \rightarrow C_2 \wedge S_1 \\ C_2 \rightarrow N_2 \wedge S_0 \end{array}$$



信号互斥例子(Mutual Exclusion)

Specification
(System Property)

Specification – Desirable Property

*No matter where you are there is
always a way to get to the initial state*

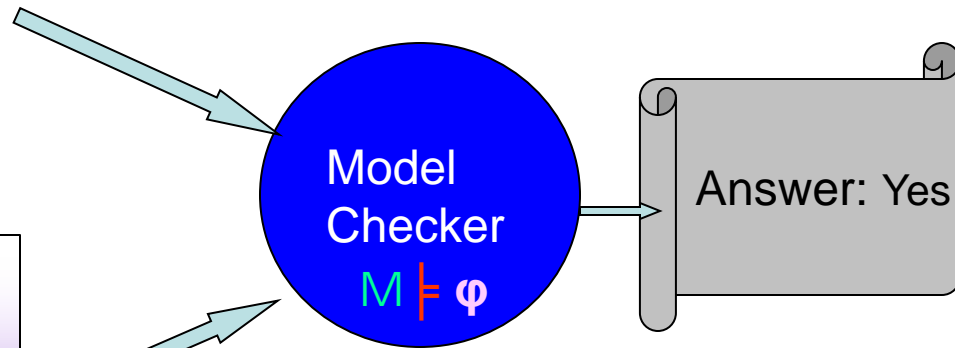
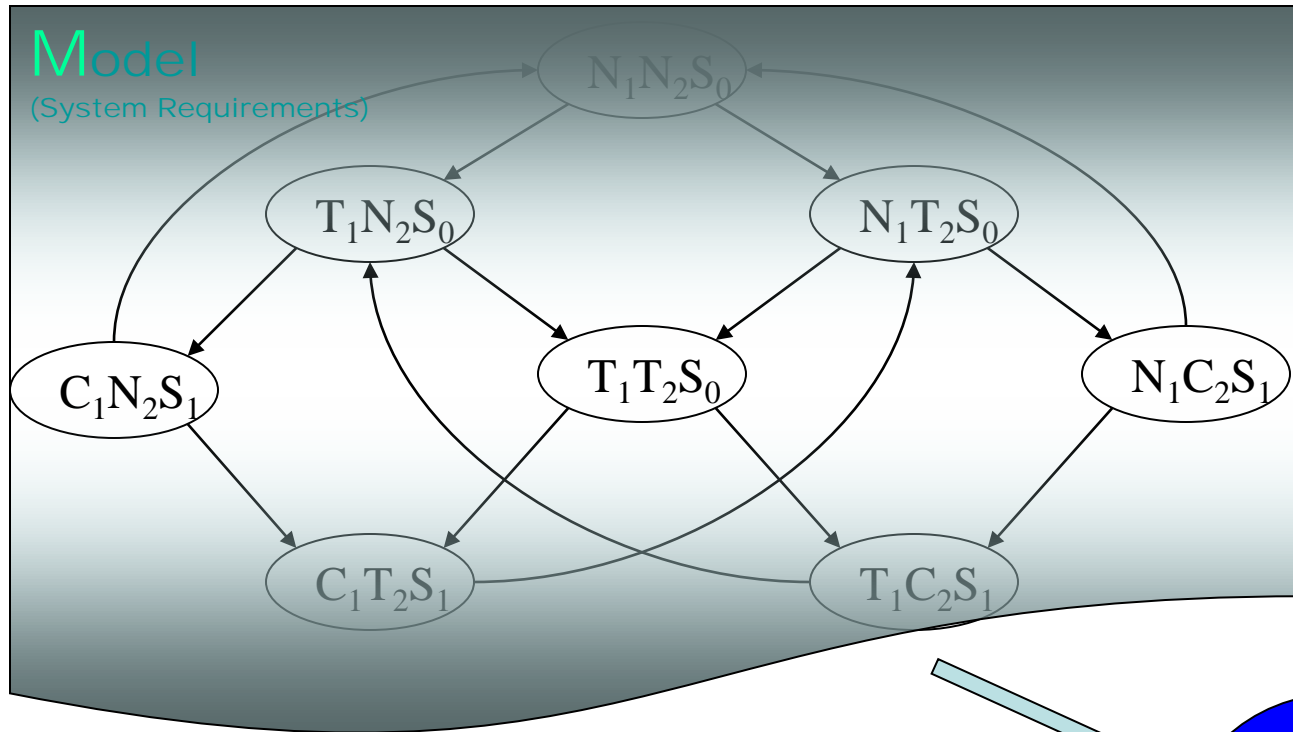
State Machine $\models \text{AG EF} (\mathbf{N}_1 \wedge \mathbf{N}_2 \wedge \mathbf{S}_0)$

CTL (Computation Tree Logic)

$M \models \varphi$

- Quantifiers over paths
 - **A** φ – All: φ has to hold on all paths starting from the current state.
 - **E** φ – Exists: there exists at least one path starting from the current state where φ holds.
- Path-specific quantifiers
 - **X** φ – Next: φ has to hold at the next state (this operator is sometimes noted **N** instead of **X**).
 - **G** φ – Globally: φ has to hold on the entire subsequent path.
 - **F** φ – Finally: φ eventually has to hold (somewhere on the subsequent path).

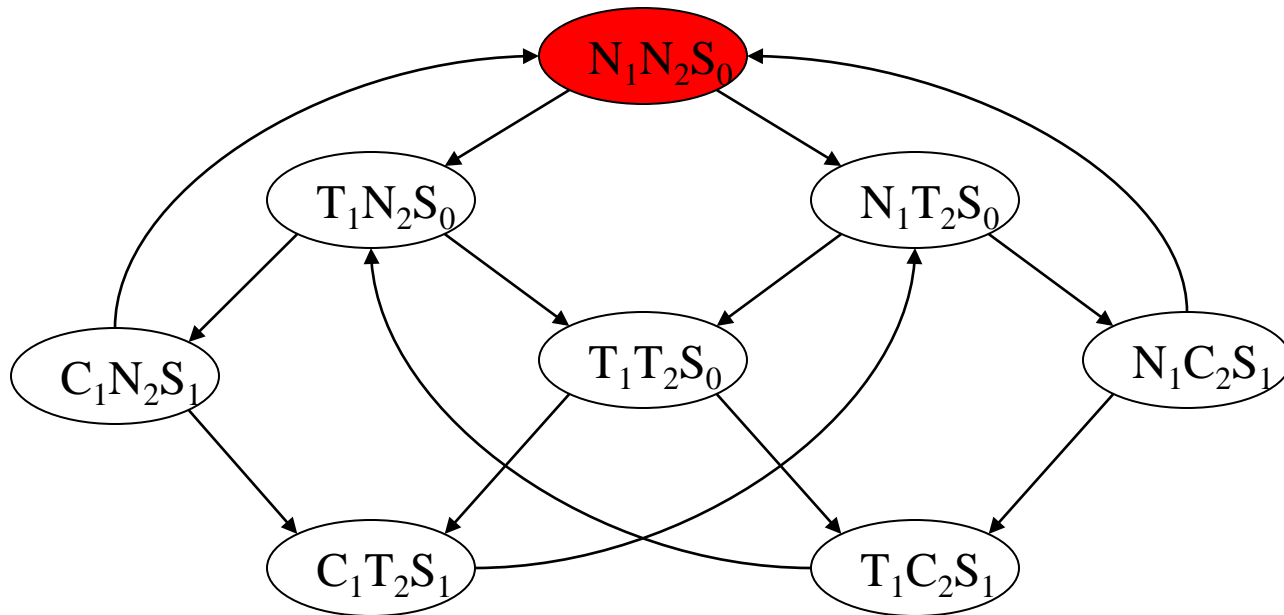
信号互斥例子(Mutual Exclusion)



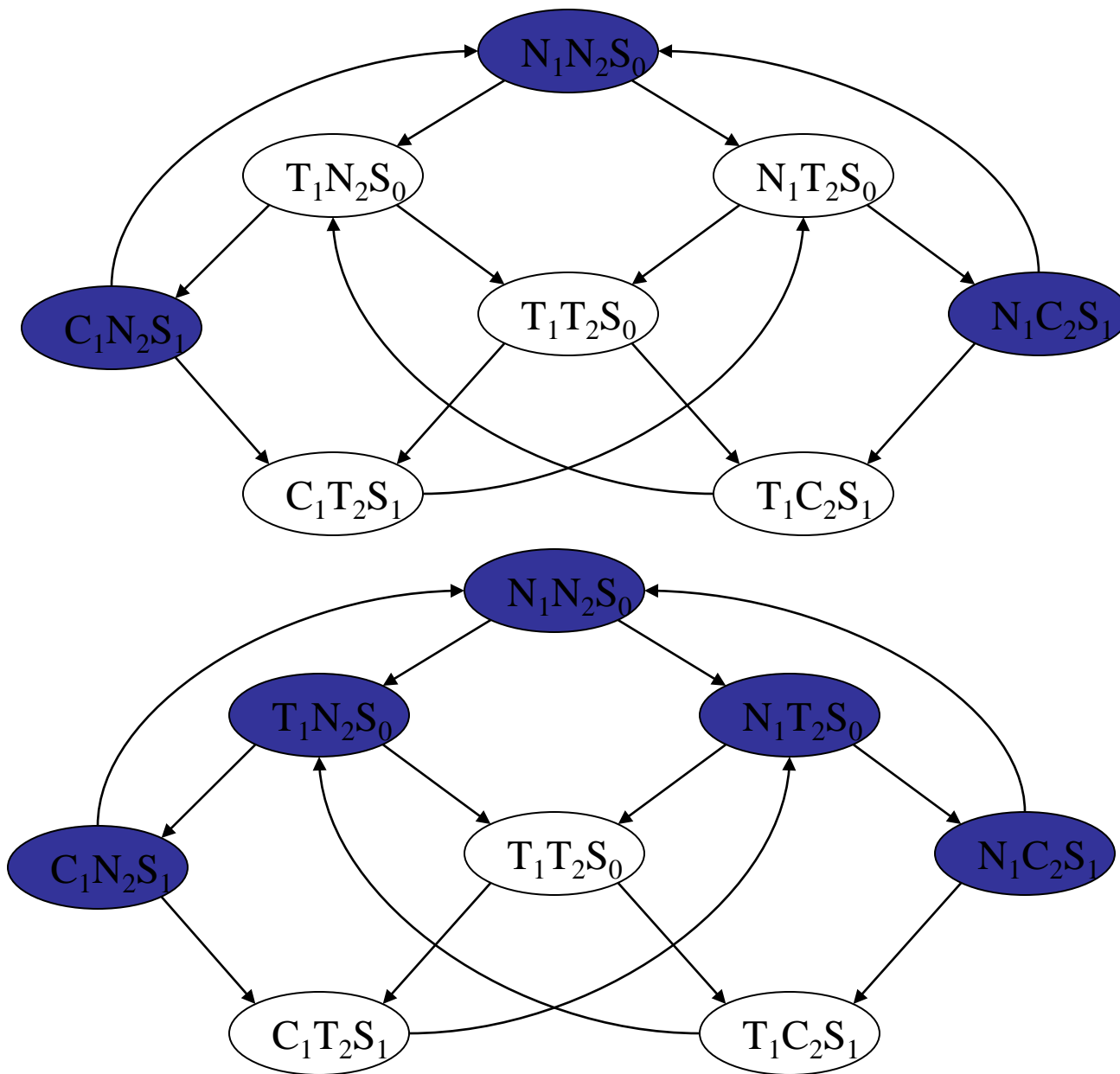
信号互斥例子(Mutual Exclusion)

Answer: Yes

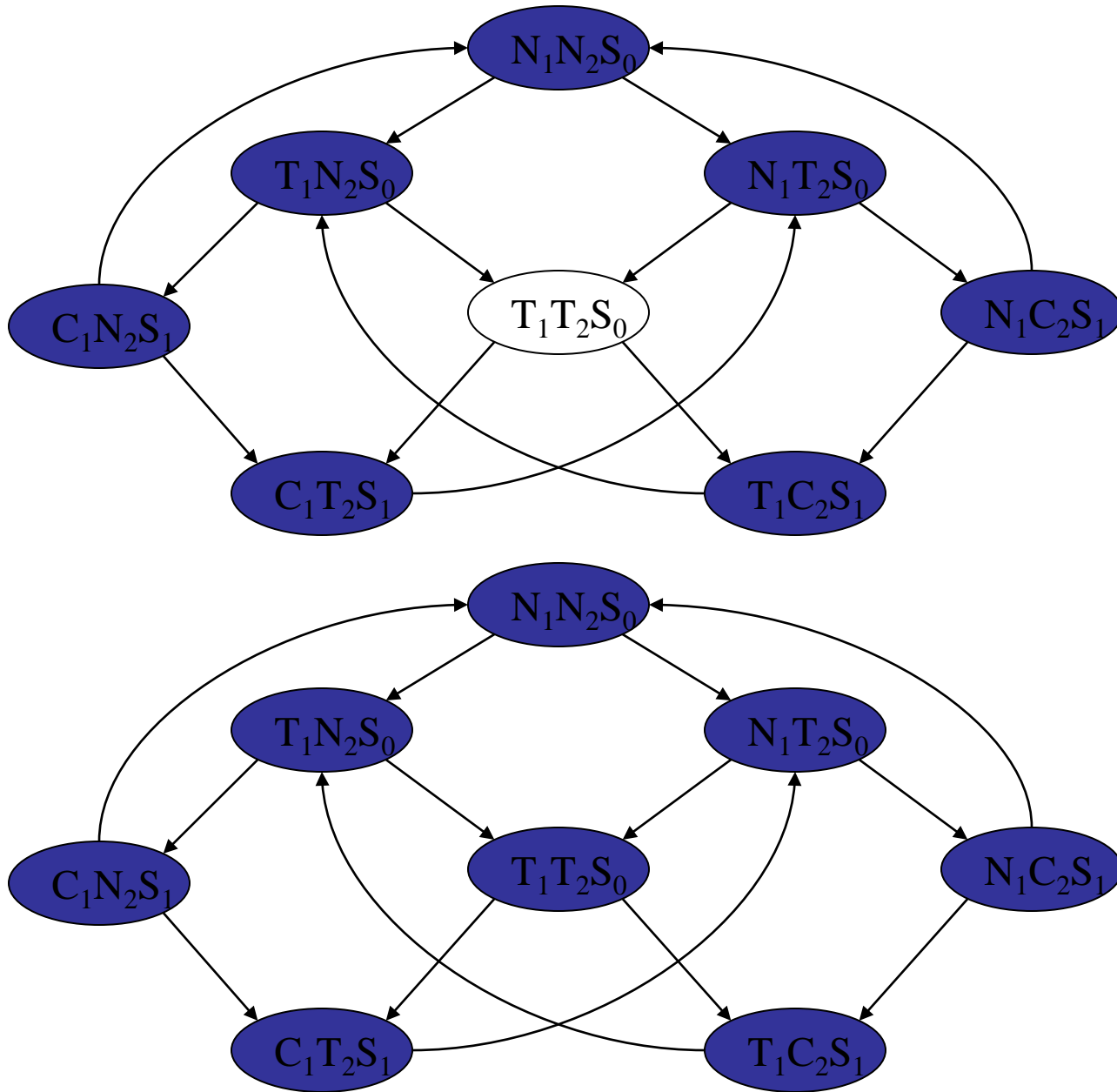
A Proof: *For All* possible behaviors



信号互斥例子(Mutual Exclusion)



信号互斥例子(Mutual Exclusion)



信号互斥例子(Mutual Exclusion)

Specification – Desirable Property

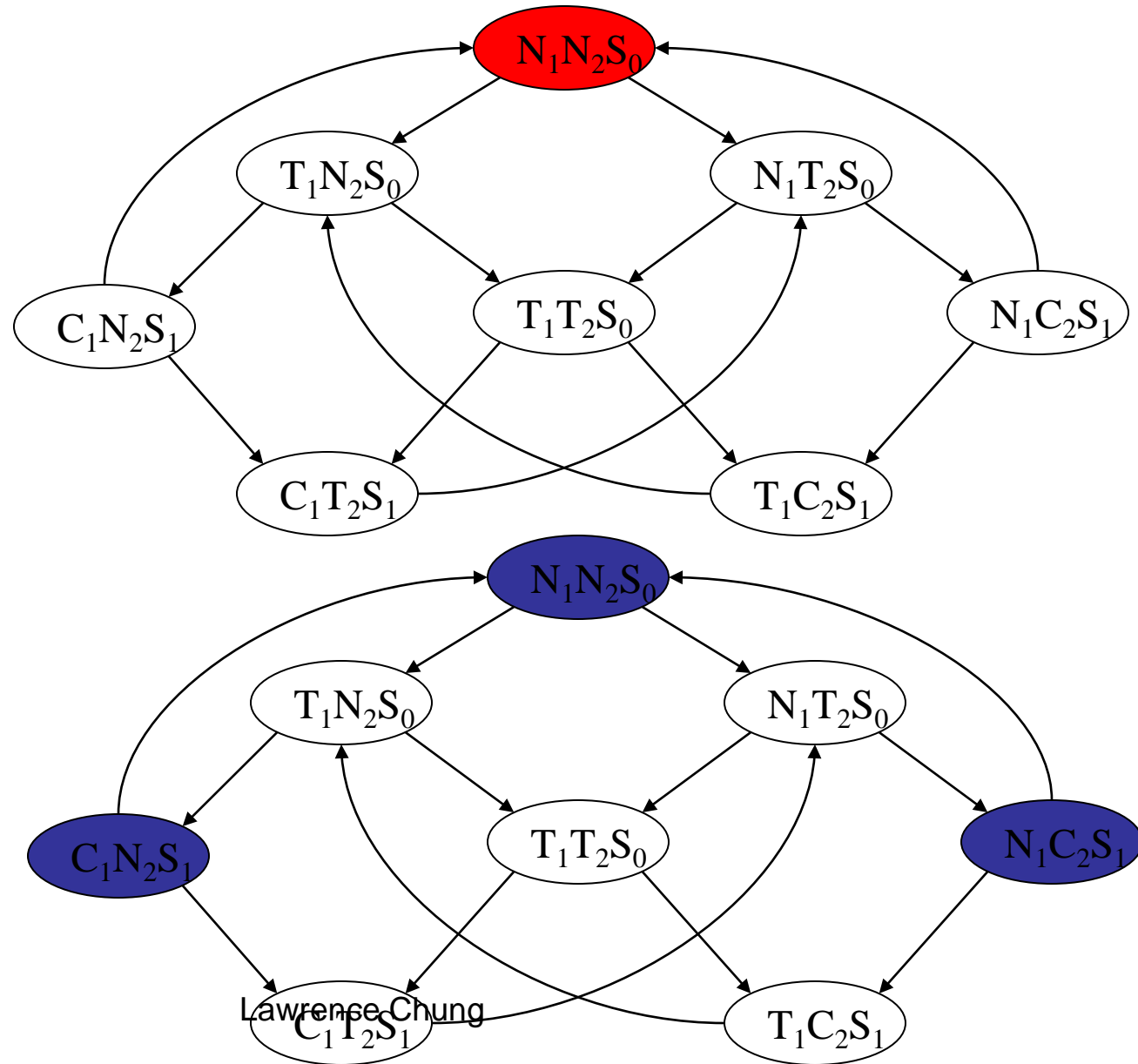
*No matter where you are there is
no way to get to the initial state*

$$\neg M \models \text{AG EF } (N_1 \wedge N_2 \wedge S_0)$$

信号互斥例子(Mutual Exclusion)

Answer: No

Counterexample



模型形式化方法

- 系统模型：状态机及其扩展
- Kripke Structures

Let AP be a set of labels — i.e., a set of atomic propositions such as Boolean expressions over variables, constants, and predicate symbols.

A Kripke structure is a 4-tuple, $M = (S, I, R, L)$:

- ▶ a finite set of states, S ,
- ▶ a set of initial states, $I \subseteq S$,
- ▶ a transition relation, $R \subseteq S \times S$ where $\forall s \in S, \exists s' \in S$ such that $(s, s') \in R$,
- ▶ a labeling function, L , from states to the power set of atomic propositions, $L : S \rightarrow 2^{AP}$.



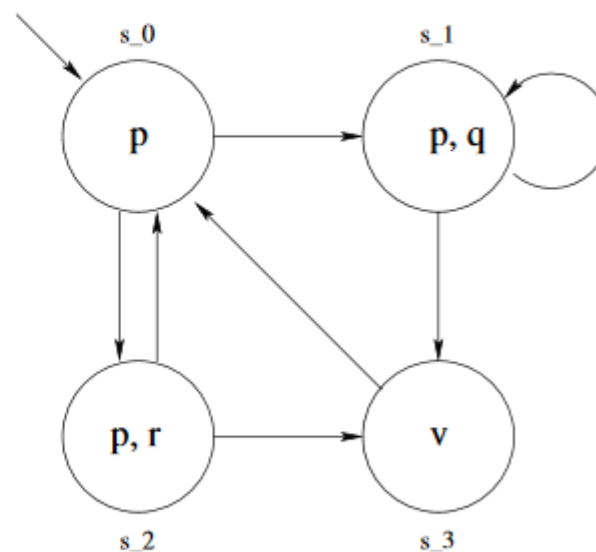
Kripke Structures例子

$$S = \{s_0, s_1, s_2, s_3\}$$

$$I = \{s_0\}$$

$$R = \left\{ \begin{array}{l} \{s_0, s_1\} \\ \{s_0, s_2\} \\ \{s_1, s_1\} \\ \{s_1, s_3\} \\ \{s_2, s_0\} \\ \{s_2, s_3\} \\ \{s_3, s_0\} \end{array} \right\}$$

$$L = \left\{ \begin{array}{l} \{s_0, \{p\}\} \\ \{s_1, \{p, q\}\} \\ \{s_2, \{p, r\}\} \\ \{s_3, \{v\}\} \end{array} \right\}$$



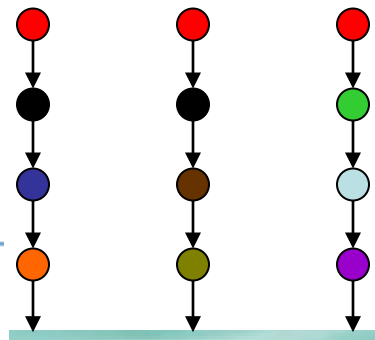
属性描述方法

• 时序逻辑(Temporal Logic)

- 描述事件的时序属性
- “Always”, “Eventually”

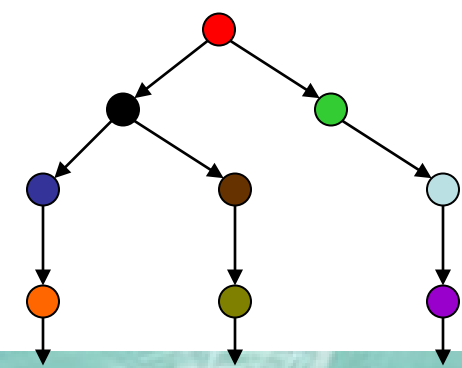
❖ Linear Time

- Every moment has a unique successor
- Infinite sequences (words)
- Linear Time Temporal Logic (LTL)



❖ Branching Time

- Every moment has several successors
- Infinite tree
- Computation Tree Logic (CTL)



模型检测方法和工具：TLA+

- **TLA(Temporal Logic of Actions)**
 - 一种描述LTL变体的规范(Specification)
- **TLA+: A TLA IDE Model Checking Tool**
 - 模型检测工具
 - 可定义状态机、可描述待验证的属性
 - 可自动推理验证属性是否为真；如为假，则自动给出反例
 - 图灵奖获得者Lamport设计
- 主页：
<https://lampport.azurewebsites.net/tla/tla.html>



TLA+例子(1)

- **Die Hard Problem**

- 有两个水壶：3 gallons, 5 gallons
- 一个水龙头可以出水
- 水壶可以接满水
- 水壶之间可以倾倒水

- **问题：采用什么方法使得大水壶里有且仅有4 gallons水？可行吗？**



TLA+例子(2)

- 模型：状态机定义
- Actions
 - 水壶灌满水
 - 水壶倒空
 - 水壶间倒水

```
VARIABLES small, big

Init == /\ big = 0 /\ small = 0

FillSmall == /\ small' = 3 /\ big' = big

FillBig == /\ big' = 5 /\ small' = small

EmptySmall == /\ small' = 0 /\ big' = big

EmptyBig == /\ big' = 0 /\ small' = small

SmallToBig == IF big + small <= 5
                THEN /\ big' = big + small
                    /\ small' = 0
                ELSE /\ big' = 5
                    /\ small' = small - (5 - big)

BigToSmall == IF big + small <= 3
                THEN /\ big' = 0
                    /\ small' = big + small
                ELSE /\ big' = small - (3 - big)
                    /\ small' = 3

Next == \/ FillSmall
        \/ FillBig
        \/ EmptySmall
        \/ EmptyBig
        \/ SmallToBig
        \/ BigToSmall
```



TLA+例子(3)

- 定义属性: $\text{Big4} == \text{big} \neq 4$
 - big变量always不等于4
- TLA+自动验证Big4属性
 - 如为真, 说明不存在一种方法使得大水壶中水量为4
 - 否则, 给出一种方案使得big=4



TLA+例子(4)

```
TLA Module
4  VARIABLES small, big
5
6  TypeOK == /\ small \in 0..3
7             /\ big \in 0..5
8
9  Big4 == big /= 4
10
11 Init == /\ big = 0 /\ small = 0
12
13 FillSmall == /\ small' = 3 /\ big' = big
14
15 FillBig == /\ big' = 5 /\ small' = small
16
17 EmptySmall == /\ small' = 0 /\ big' = big
18
19 EmptyBig == /\ big' = 0 /\ small' = small
20
21 SmallToBig == IF big + small =< 5
22                THEN /\ big' = big + small
23                     /\ small' = 0
24                ELSE /\ big' = 5
25                     /\ small' = small - (5 - big)
26
27 BigToSmall == IF big + small =< 3
28                THEN /\ big' = 0
29                     /\ small' = big + small
30                ELSE /\ big' = small - (3 - big)
31                     /\ small' = 3
32
33 Next == \/ FillSmall
34         \/ FillBig
```

Invariant Big4 is violated.

Error-Trace Exploration

Error-Trace

Name	Value
■ small	0
▲ ▲ <Action line 27, col 15 to line 31, col 33 o	State (num = 3)
■ big	2
■ small	3
▲ ▲ <Action line 17, col 15 to line 17, col 43 o	State (num = 4)
■ big	2
■ small	0
▲ ▲ <Action line 27, col 15 to line 31, col 33 o	State (num = 5)
■ big	0
■ small	2
▲ ▲ <Action line 15, col 12 to line 15, col 42 o	State (num = 6)
■ big	5
■ small	2
▲ ▲ <Action line 27, col 15 to line 31, col 33 o	State (num = 7)
■ big	4
■ small	3

其余模型检测方法和工具

- 自定义状态机
- 直接使用LTL和CTL的语法
- 自己设计状态空间搜索算法

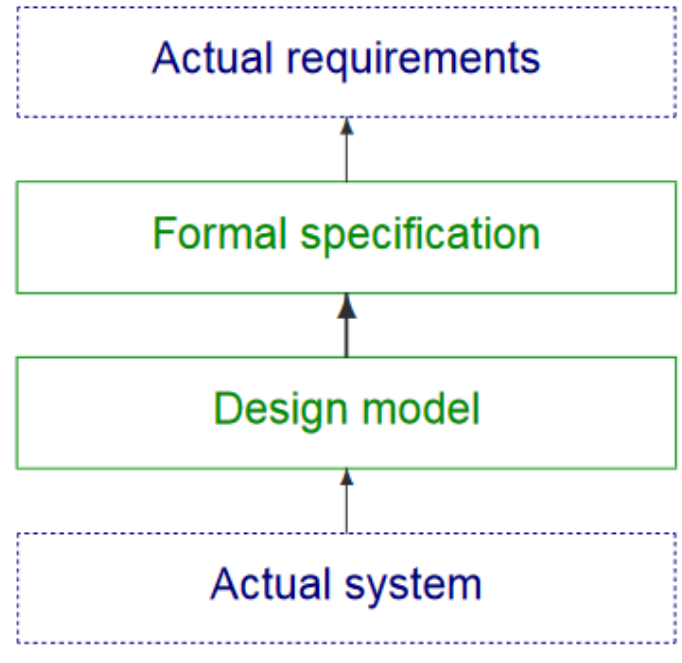
- 其它工具
 - SPIN: LTL
 - LTSmin: LTL, CTL*(CTL extension)
 - 具体查看
https://en.wikipedia.org/wiki/List_of_model_checking_tools



演绎推理

- 定理证明和推理

- 设计模型：从真实代码中抽象出来
- 形式化需求规约：真实需求的形式化表示
- 证明：设计模型是否符合形式化需求规约



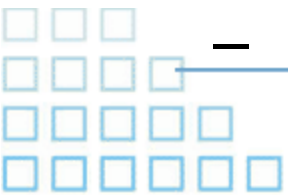
如何形式化？

- 形式化设计模型和需求规约
- 逻辑方法
 - 命题逻辑(布尔代数, Proposition Logic) PL
 - 一阶逻辑(First-order Logic) FOL
 - 高阶逻辑(High-order Logic) HOL
- 形式化语言的表达能力越强，后续的证明过程自动化的难度越大



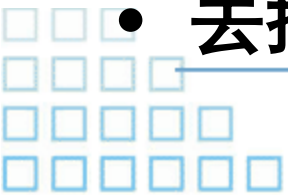
命题逻辑

- 原子命题 p, q
- 命题公式: $p \vee \neg p, \neg(p \wedge q) \Leftrightarrow \neg p \vee \neg q$
- 证明过程是可满足性问题(SAT)
 - 第一个被证明的NP完全问题
 - 除非 $P=NP$, 不存在线性的自动推理方法
- 实际算法: 足够好
 - Davis-Putnam-Loveland-Logemann (DPLL)
 - Stalmarck's method



一阶逻辑

- 变量和函数: $x, y, f(x)$
- 算术公式: $x+1 < 2y, R(f(x),y)$
- 量词: $\forall x \exists y (f(x))$
- 证明过程是FOL验证问题
 - 不可判定问题, 半可判定
 - 为真(即定理), 存在算法最终可验证; 为假, 可能一直计算无法判断
 - 不存在有效的验证算法
- 去掉量词的一阶逻辑: 存在有效验证算法SMT



高阶逻辑

- 在一阶逻辑基础上，量词可作用于谓词和函数：
 - $\forall P. P(0) \wedge (\forall n. P(n) \Rightarrow P(n+1)) \Rightarrow \forall n. P(n)$
 - 自然数：同构于二阶逻辑公理系统
- 证明过程是HOL验证问题
 - 甚至不是半可判定的
 - 不存在有效的验证算法
- 去掉量词的一阶逻辑： 存在有效验证算法SMT



类型理论(Type Theory)

- **类型理论**
 - 一种形式化系统：包括类型以及类型上的操作
 - 典型代表：Calculus of (inductive) constructions (CoC, CIC)，即typed lambda calculus
- **HOL的量词可作用于谓词和函数**
 - 可看作是类型以及高阶的类型
 - CoC可用于描述HOL
- 一般认为，CoC和HOL的表达能力差不多，在实际使用上不严格区分



软件交互式定理证明

- **HOL甚至FOL不存在有效的自动推理工具**
 - 命题逻辑：除非 $P=NP$ ，不在线性的自动推理方法
 - FOL：半可判定
 - HOL：甚至不是半可判定
- **交互式的定理证明(Interactive Theorem Prover)**
 - 人工辅助的方式进行快速推理
 - 使之成为实用且有效的定理证明工具

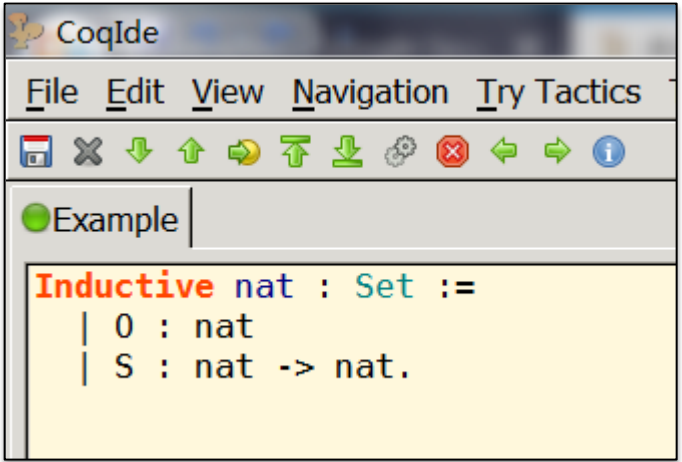


交互式定理证明工具

- **Coq工具**: <https://coq.inria.fr/>
 - 基于CoC, 可用于描述HOL和Typed函数式编程
- **HOL工具**: <https://hol-theorem-prover.org/>
 - 描述HOL
- **Isabelle工具**: <https://isabelle.in.tum.de/>
 - 描述HOL
- **即使最易用的证明器, 也不是一件容易的事情**
 - 需要人工进行形式化
 - 需要人工交互和干预
- **原则: 让能够自动化的, 使它自动化**

Coq工具例子(1)

- **Peano算术公理系统：两条核心公理**
 - 0 is a natural number.
 - For every natural number n, (S n) is a natural number.



```
CoqIde
File Edit View Navigation Try Tactics
[Icons]
● Example
Inductive nat : Set :=
| 0 : nat
| S : nat -> nat.
```

- 0 is 0
- 1 is (S 0) = (1 + 0)
- 2 is (S (S 0)) = (1 + (1 + 0))
- 3 is (S (S (S 0))) = (1 + (1 + (1 + 0)))
- ...



Coq工具例子(2)

- 定义算术加法: 递归定义函数

```
Fixpoint plus (n m:nat) : nat :=  
  match n with  
  | 0 => m  
  | S p => S (p + m)  
  end  
where "n + m" := (plus n m) : nat_scope.
```

- 2+3=? 手动推理过程:

- (plus (S (S O)) (S (S (S O))))
- (S (plus (S O) (S (S (S O)))))
- (S (S (plus O (S (S (S O)))))
- (S (S (S (S (S O)))))



Coq工具例子(3)

- 定理自动证明

```
CoqIde
File Edit View Navigation Try Tactics Templates Queries Tools Compile Windows H
Example
Inductive nat : Set :=
| 0 : nat
| S : nat -> nat.

Fixpoint plus (n m:nat) : nat :=
  match n with
  | 0 => m
  | S p => S (p + m)
  end
where "n + m" := (plus n m) : nat_scope.

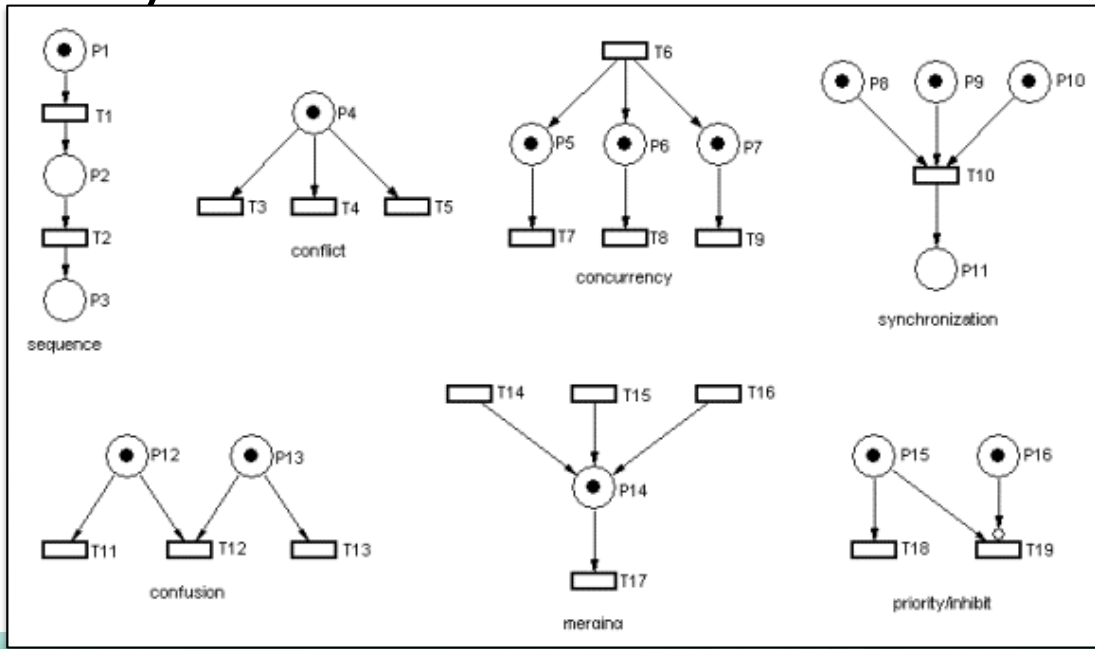
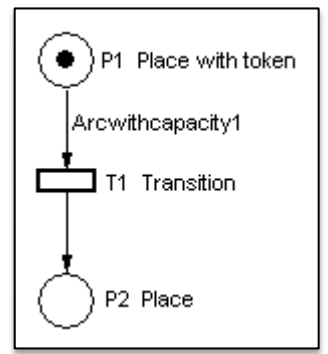
Theorem plus_2_3 : (S (S 0)) + (S (S (S 0))) = (S (S (S (S (S 0))))).
Proof.
  simpl.
  exact (eq_refl 5).
Qed
```

```
1 subgoal
_____ (1/1)
S (S (S (S (S 0)))) = S (S (S (S (S 0))))
```

其他方法：Petri Nets

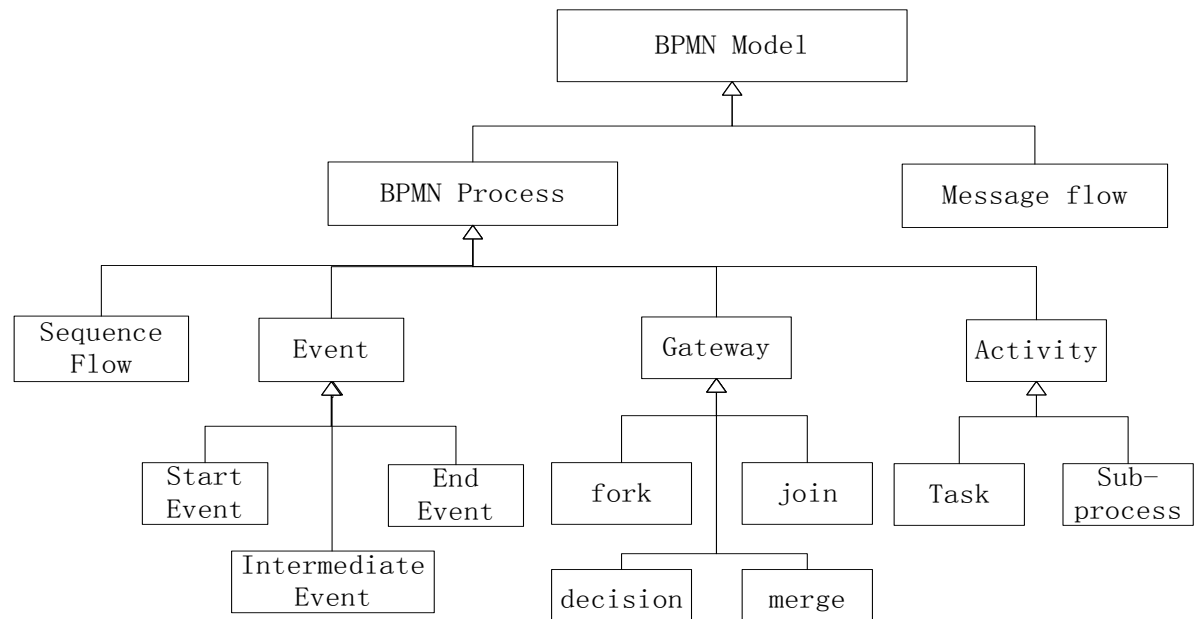
- **Petri Net:**

- a collection of directed arcs connecting places(库所) and transitions(变迁)
- Places may hold tokens


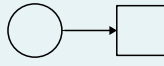

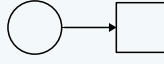



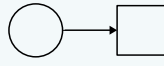
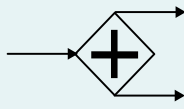
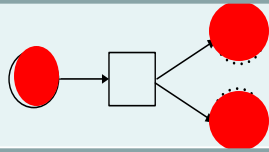
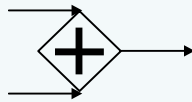
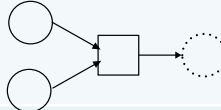
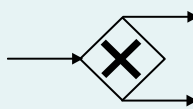
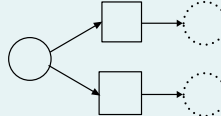
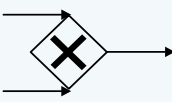
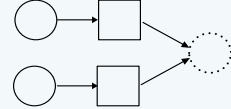


基于Petri网的组合服务验证

- ◆ 组合服务用业务流程模型BPMN表示
- ◆ 一个Core BPMN Model是一个二元组 $CBM = (Q, FM)$ ，其中：
 - Q 是核心BPMN流程CBP的集合
 - FM 是流程之间的消息流message Flow的集合
- ◆ 一个Core BPMN Process是一个五元组 $CBP = (O, T, E, G, F)$

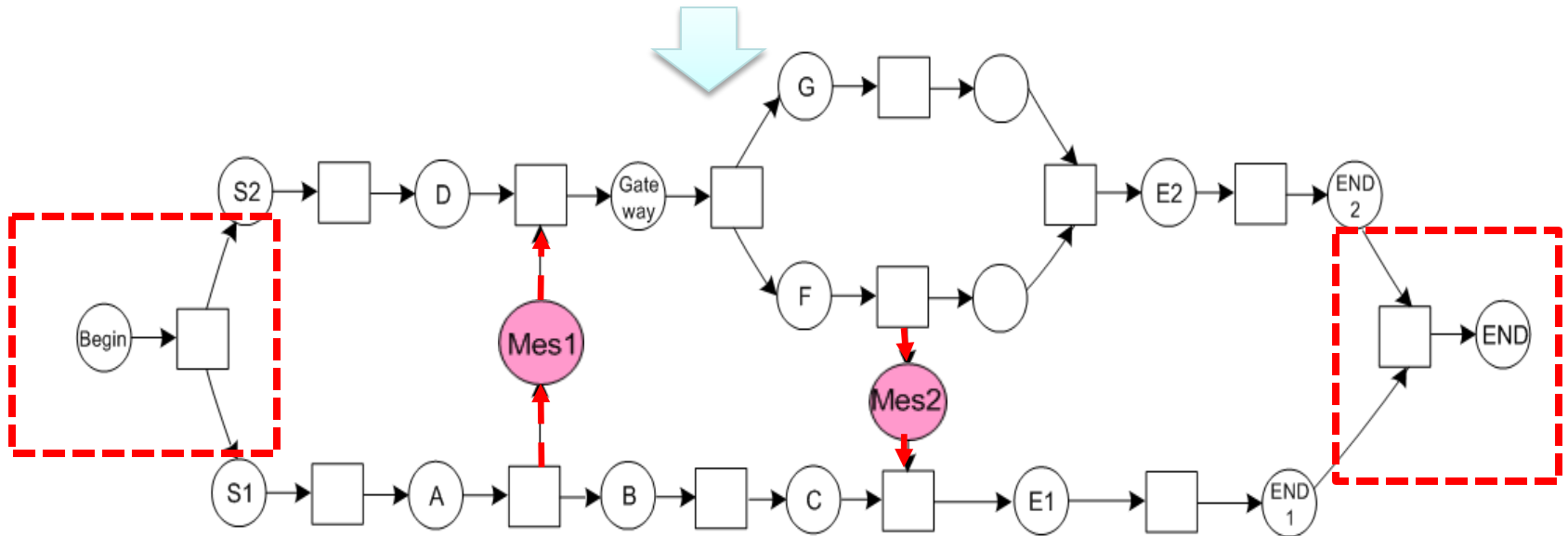
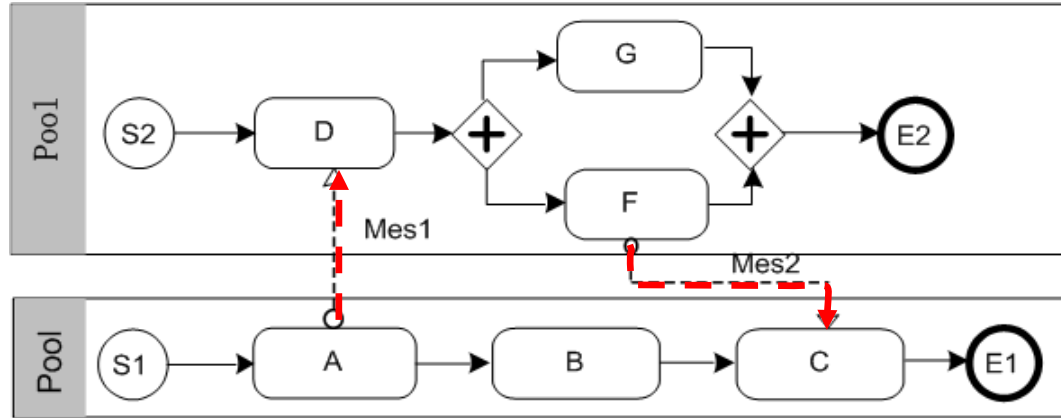


BPMN2Petri net映射

BPMN图元 — Flow Objects		Petri net	
event	start		
	intermediate		
	end		
	task		
gateway	fork		
	join		
	decision		
	merge		

BPMN2Petri net映射

分析示例



模型安全性验证

◆ 将Petri网模型转换为有向图 $G(V, E)$

- 顶点集 $V = \{M(p_0, p_1, \dots, p_n)\}$ 是模型的状态集合
- 有向弧 E 记录可执行变迁及其引起的状态迁移

◆ 验证属性

- 可达性
 - 图搜索
- 死锁
 - 查找可达图的叶子节点。若叶子节点的状态标识中存在 token 等于 1 的库所，且该库所不是结束库所
- 环路
 - 验证方法：环路搜索算法

Q&A

