時相論理(temporal logic)

様相論理(modal logic)の一種

命題の真理値の時間に沿った変化を記述

　　LTL　線形時相論理

　　CTL　分岐的時相論理

プログラムのlivenessやsafety properties等を

記述するのに便利

モデル検査(Model Checking)に有用

　　公理系もある

LTL
(linear-time temporal logic)

Syntax

p ∈ Atoms 原始命題 (propositional atoms)

ϕ::=

T | ⊥ | p | (¬ ϕ) | (ϕ ∧ ϕ) | (ϕ ∨ ϕ) | (ϕ ⊃ ϕ)

| (X ϕ)	neXt
| (F ϕ)	some Future state
| (G ϕ)	Globally (all future states)
| (ϕ U ϕ)	Until
| (ϕ W ϕ)	Weak until
| (ϕ R ϕ)	Release

Semantics (意味)

Transition System

transition system (model) M=(S, →, L)

S : 状態 (state)の集合

→ : S→S ：遷移関係 (transition relation)

L : S → P(Atoms)

: ラベル関数 (labelling function)

Atoms: 原始論理式(atomic formulas)の集合

計算列 (computation path)

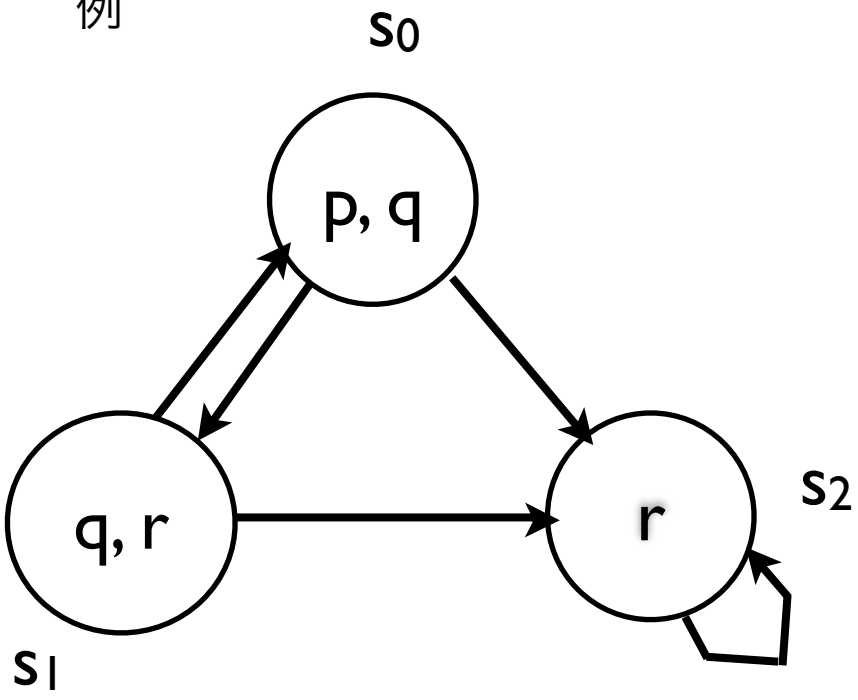
s1, s2, … ただしsi∈S, si→ si+1

π=s1→s2→…のように表す

πi: siから始まるπの接尾辞(suffix)

例: π3=s3→s4→…

例



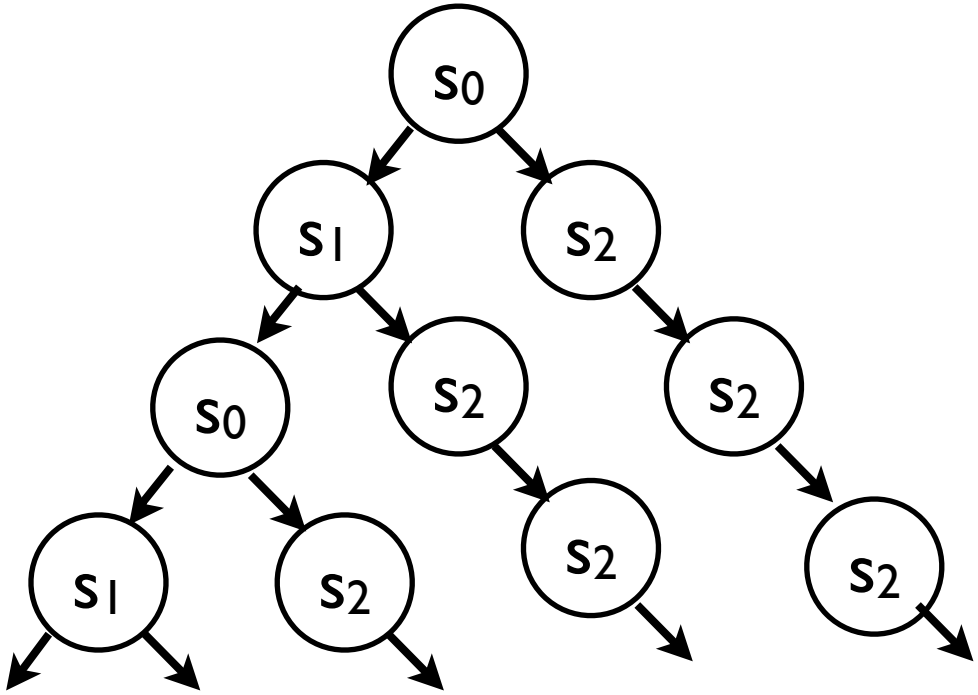
$s_0 \to s_2 \to s_2 \to s_2 \to s_2 \to s_2 \to \ldots$

$s_0 \to s_1 \to s_2 \to s_2 \to s_2 \to s_2 \to \ldots$

$s_0 \to s_1 \to s_0 \to s_2 \to s_2 \to s_2 \to \ldots$

$s_0 \to s_1 \to s_0 \to s_1 \to s_2 \to s_2 \to \ldots$

...



Semantics (意味)

$M=(S, \rightarrow, L)$, $\pi=s1\rightarrow s2\rightarrow\ldots$

: 充足関係(satisfaction relation)を定義

$\pi \models T$

$\pi \not\models \bot$

$\pi \models p$ iff $p \in L(s1)$

$\pi \models \neg\phi$ iff $\pi \not\models \phi$

$\pi \models \phi1 \wedge \phi2$ iff $\pi \models \phi1$ and $\pi \models \phi2$

$\pi \models \phi1 \vee \phi2$ iff $\pi \models \phi1$ or $\pi \models \phi2$

$\pi \models \phi1 \supset \phi2$ iff $\pi \models \phi2$ whenever $\pi \models \phi1$

$\pi \models X\phi$ iff $\pi_2 \models \phi$

$\pi \models G\phi$ iff $\pi_i \models \phi$ for every $i \geq 1$

$\pi \models F\phi$ iff $\pi_i \models \phi$ for some $i \geq 1$

$\pi \models \phi U \psi$ iff $\pi_i \models \psi$ for some $i \geq 1$ and
$\pi_j \models \phi$ for every j such that $1 \leq j < i$

$\pi \models \phi W \psi$ iff $\pi_i \models \psi$ for some $i \geq 1$ and
$\pi_j \models \phi$ for every j
such that $1 \leq j < i$
or
$\pi_k \models \phi$ for every $k \geq 1$

($\varphi W \psi \equiv (\varphi U \psi) \vee G\varphi$)

$\pi \models \phi R \psi$ iff $\pi_i \models \phi$ for some $i \geq 1$ and
$\pi_j \models \psi$ for every j such that $1 \leq j \leq i$
or
$\pi_k \models \psi$ for every $k \geq 1$

($\varphi R \psi \equiv \psi W (\varphi \vee \psi)$)

実用的な仕様表現パターン

G¬(*started*∧¬*ready*)

startedではあるがreadyではないということは

あり得ない

G(*requested*⊃F *acknowledged*)

requestがあればいずれは受理する

G F *enabled*

enabledな状態が無限回起こる

証明)背理法

有限回しか成り立たないとする.

最後にenabledが成立つ状態をs_mとおく.

s_{m+1}でもF enabledが成立つのでn>mなるs_nでもenabledが成立つ. 矛盾

F G *deadlock*

いずれdeadlockになり，それが永続的に成り立つ

G F *enabled* ⊃ G F *running*

無限回enabledならば無限回実行可能

G(*floor2*∧*directionup*∧*BottonPressed5*⊃
(*directionup* U *floor5*))

エレベータが2階にいて，上向きになっていて，

5階行きのボタンが押されたならば，

5階に到着するまで上向きのままである．

M=(S, →, L), s∈Sのとき

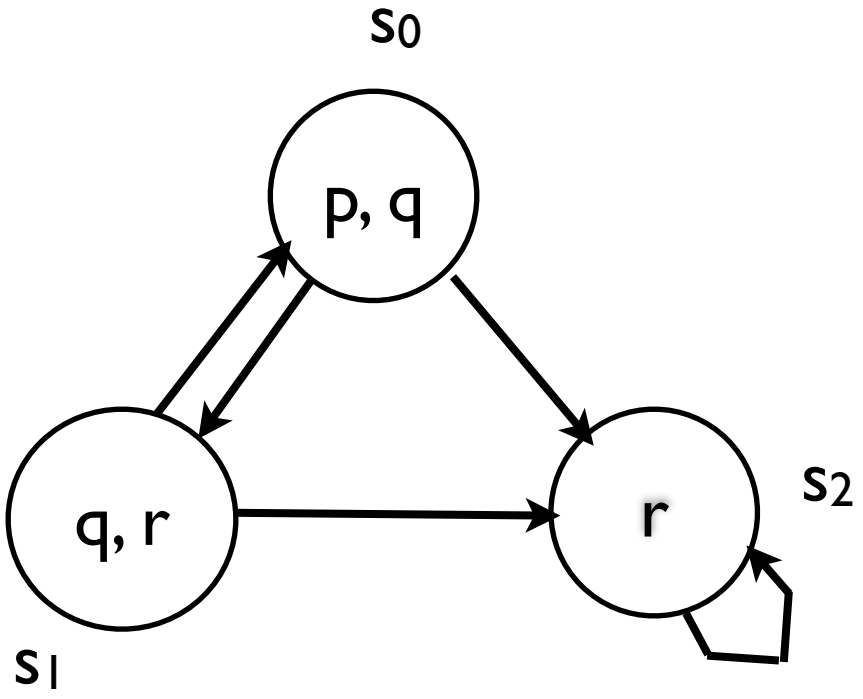
$M, s \models \phi \overset{d}{\Leftrightarrow}$ sで始まるMの全て計算列πでπ $\models \phi$

議論の文脈上Mが明らかなときは

単に $s \models \phi$ と表す.

例

- ○ $s_0 \models p \wedge q$
- ○ $s_0 \models Xr$
- × $s_0 \models X(q \wedge r)$
- ○ $s_0 \models \neg G(r \wedge q)$
- × $s_0 \models F(p \wedge r)$
- ○ $s_0 \models pUr$
- ○ $s_2 \models FGr$



どのsに対しても

$s \models F(\neg q \wedge r) \supset FGr$

演習

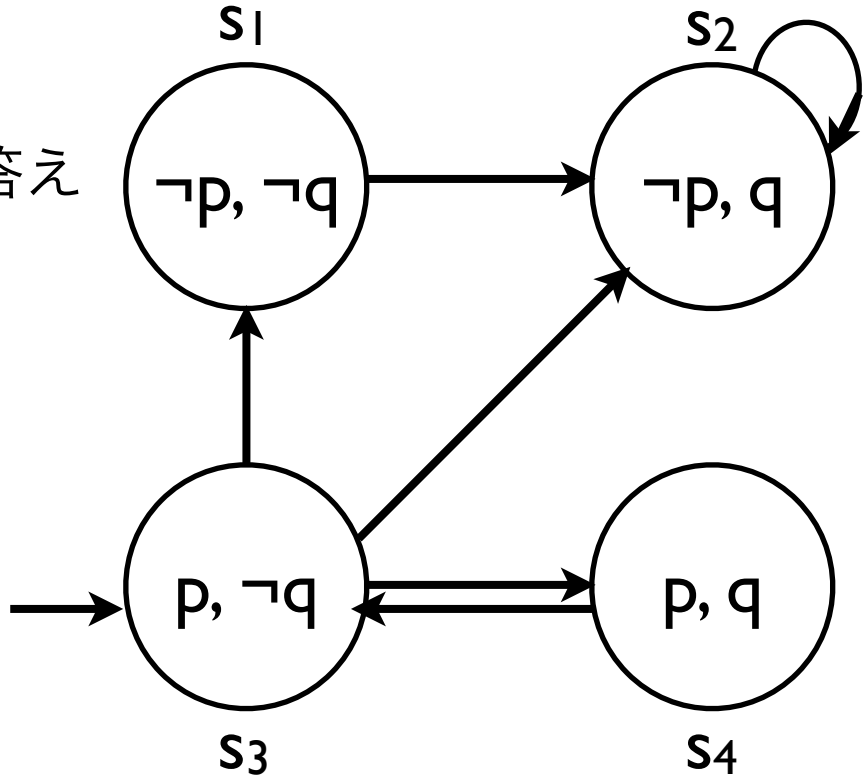
右図で与えられたモデル*M*を考える.

式(a)-(e)(各々φとする)に対して(1)(2)を答えよ.

(a) Gp　(b) pUq　(c) pUX(p∧¬q)
(d) X¬q ∧ G(¬p∨¬q)
(e) X(p∧q) ∧ F(¬p∧¬q)

(1) s_3から始まりφを満たす計算列を求めよ.

(2) M,s_3⊨φか否かを答えよ.



A model of *M*

同値なLTL論理式

定義

二つのLTL論理式φ, ψが, 全てのモデルMとM中の全ての計算列πに対して

$$\pi \models \phi \quad \text{iff} \quad \pi \models \psi$$

のとき, φとψは**意味的に同値**, または単に**同値**といい,

$$\phi \equiv \psi$$

と記す.

φがある論理式χの部分論理式であり, φ≡ψのとき, χ中のφを全てψに置き換えたものはχと同値である.

同値な式の例

$$\neg G\phi \equiv F\neg\phi$$

$$\neg F\phi \equiv G\neg\phi$$

$$\neg X\phi \equiv X\neg\phi$$

$$\neg(\phi U\psi) \equiv \neg\phi R\neg\psi$$

$$\neg(\phi R\psi) \equiv \neg\phi U\neg\psi$$

$$F(\phi\vee\psi) \equiv F\phi\vee F\psi$$

$$G(\phi\wedge\psi) \equiv G\phi\wedge G\psi$$

$$F\phi \equiv TU\phi$$

$$G\phi \equiv \bot R\phi$$

仕様表現/検証の例

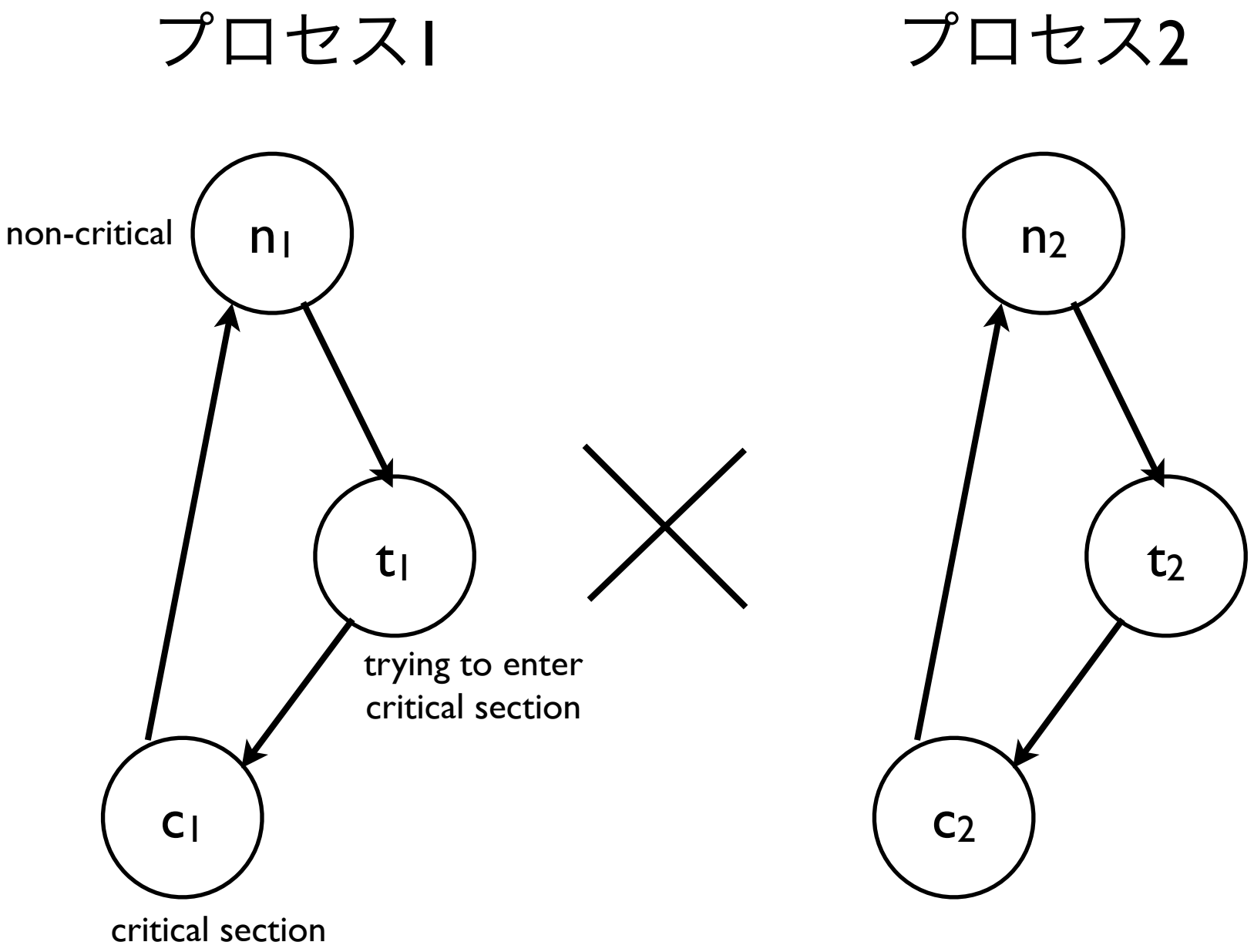
並行プログラム系の相互排除(mutual exclusion)問題を考える

Safety property : 危険領域(critical section)にアクセスしているのは常に高々1プロセス

(mutual exclusion)
Liveness property: 危険領域に(アクセスするリクエストを出せば)いずれアクセスできる

(Starvation-free/deadlock-free)
No strict sequencing: プロセスが危険領域に交互にアクセスする必要はない

Non-blocking: プロセスはいつでも危険領域にアクセスするリクエストが出せる


プロセス1
プロセス2
non-critical
n_1
t_1
trying to enter
critical section
c_1
critical section
n_2
t_2
c_2

Safety : $G\neg(c_1 \wedge c_2)$

Liveness: $G(t_1 \supset Fc_1)$

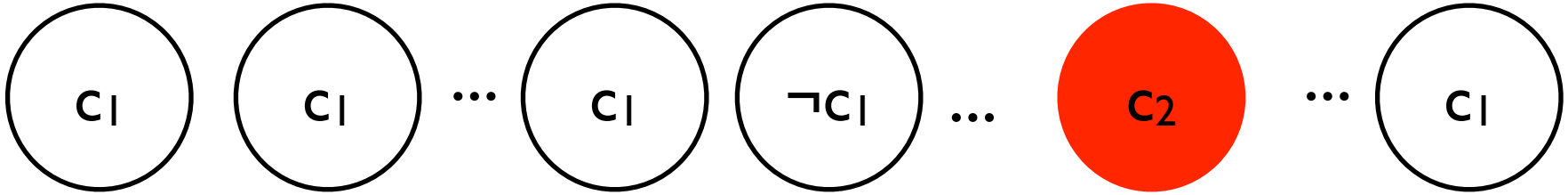
No strict sequencing:

$\neg G(c_1 \supset c_1 W(\neg c_1 \wedge \neg c_1 W c_2))$

「c_1になればそれが永久に続くか,

c_1でなくなった後c_2が起こるまで再びc_1は起こらない」

ことはない



strict seq.では

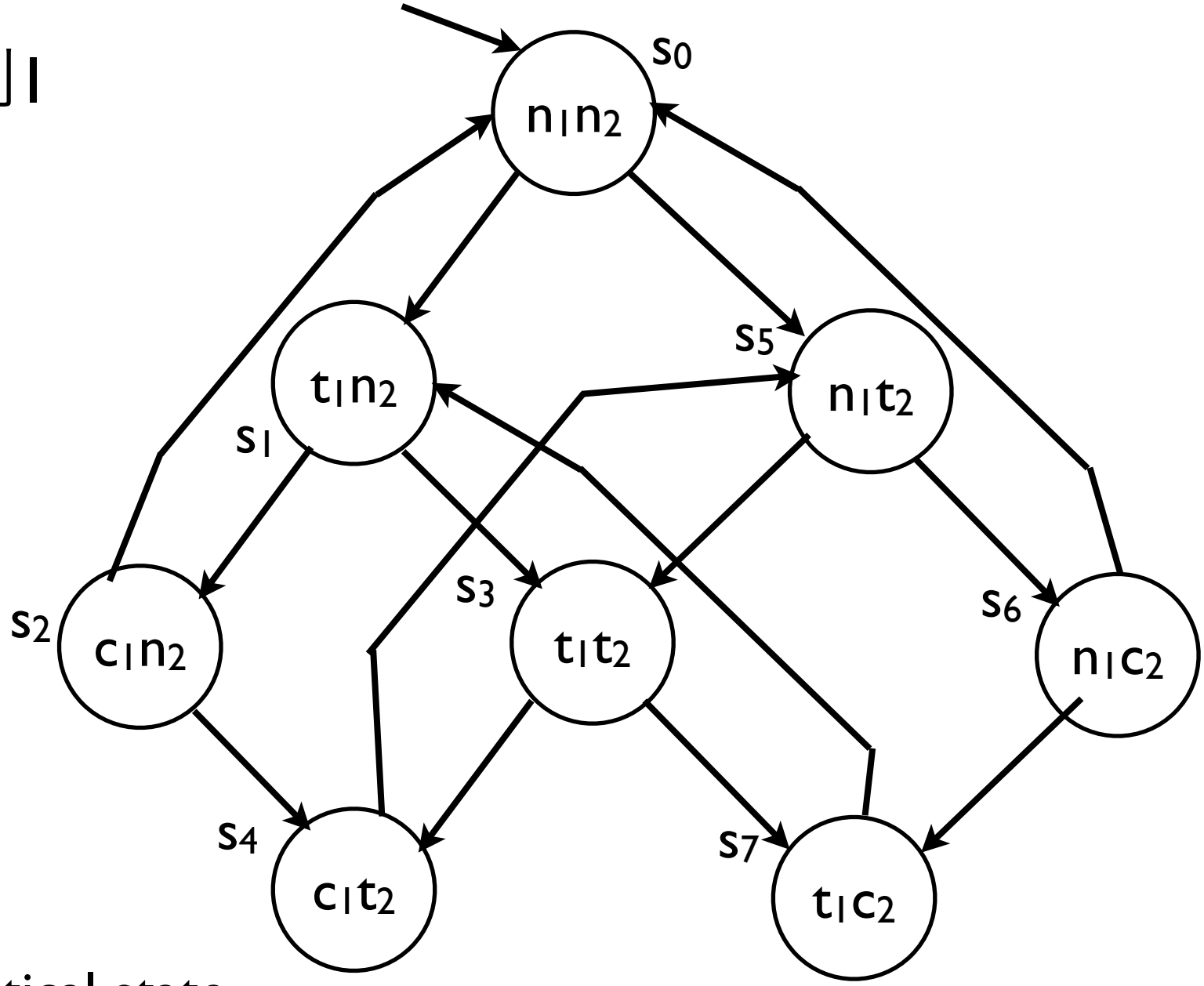
必ず起こる

Non-Blocking: 「n_1の状態のうち必ず次にt_1になるものがある」

これはLTLでは表すことが出来ない

(「ある計算列が存在して」のような表明は表せない)

例1



n: non-critical state

t: trying to enter critical state

c: critical state

○ Safety : $G\neg(c_1 \wedge c_2)$

× Liveness: $G(t_1 \supset Fc_1)$

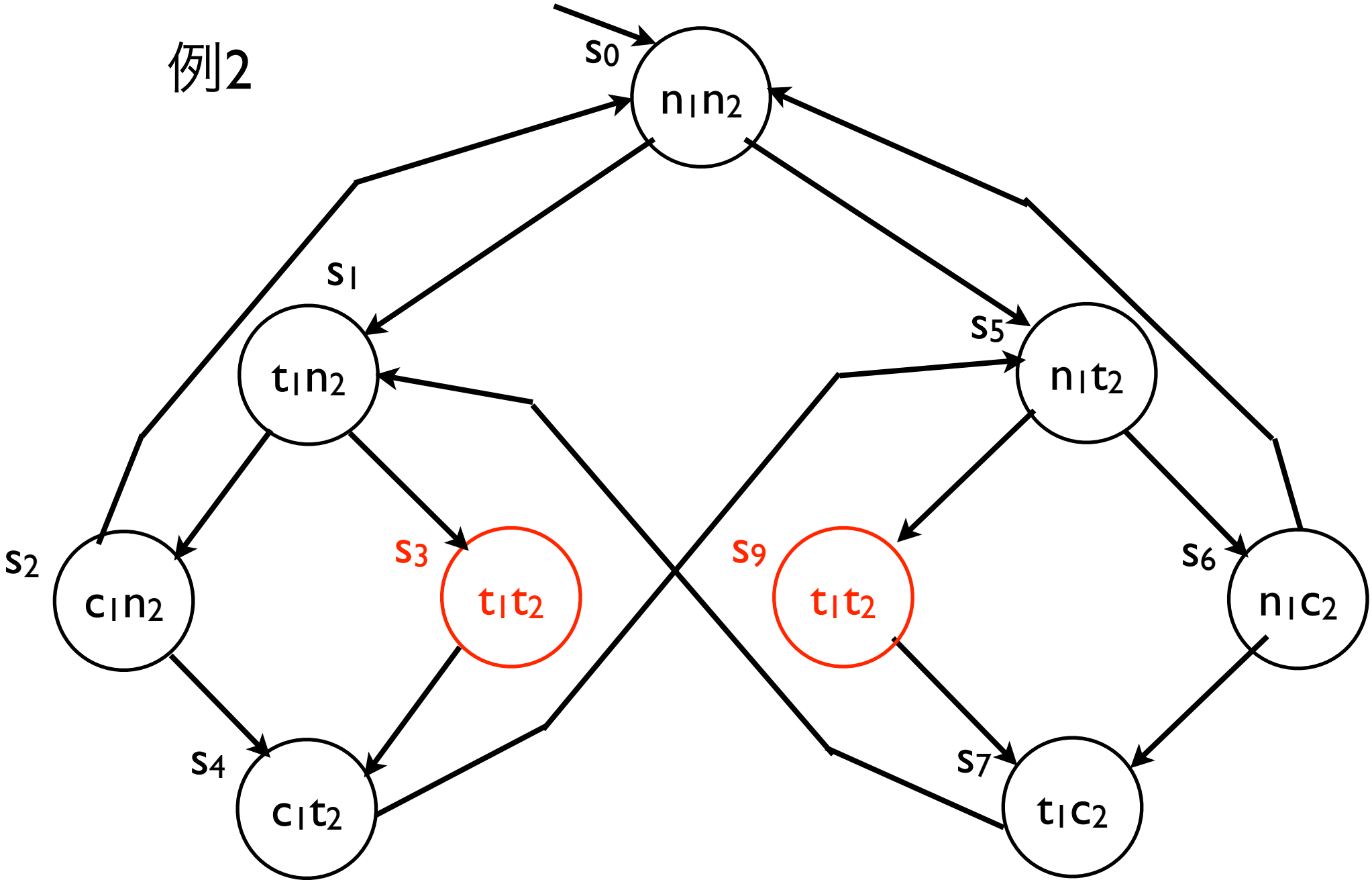
○ No strict sequencing:

$\neg G(c_1 \supset c_1 W(\neg c_1 \wedge \neg c_1 W c_2))$

c_1になればそれが永久に続くか,

c_1でなくなった後c_2が起こるまで再びc_1は起こらない

例2

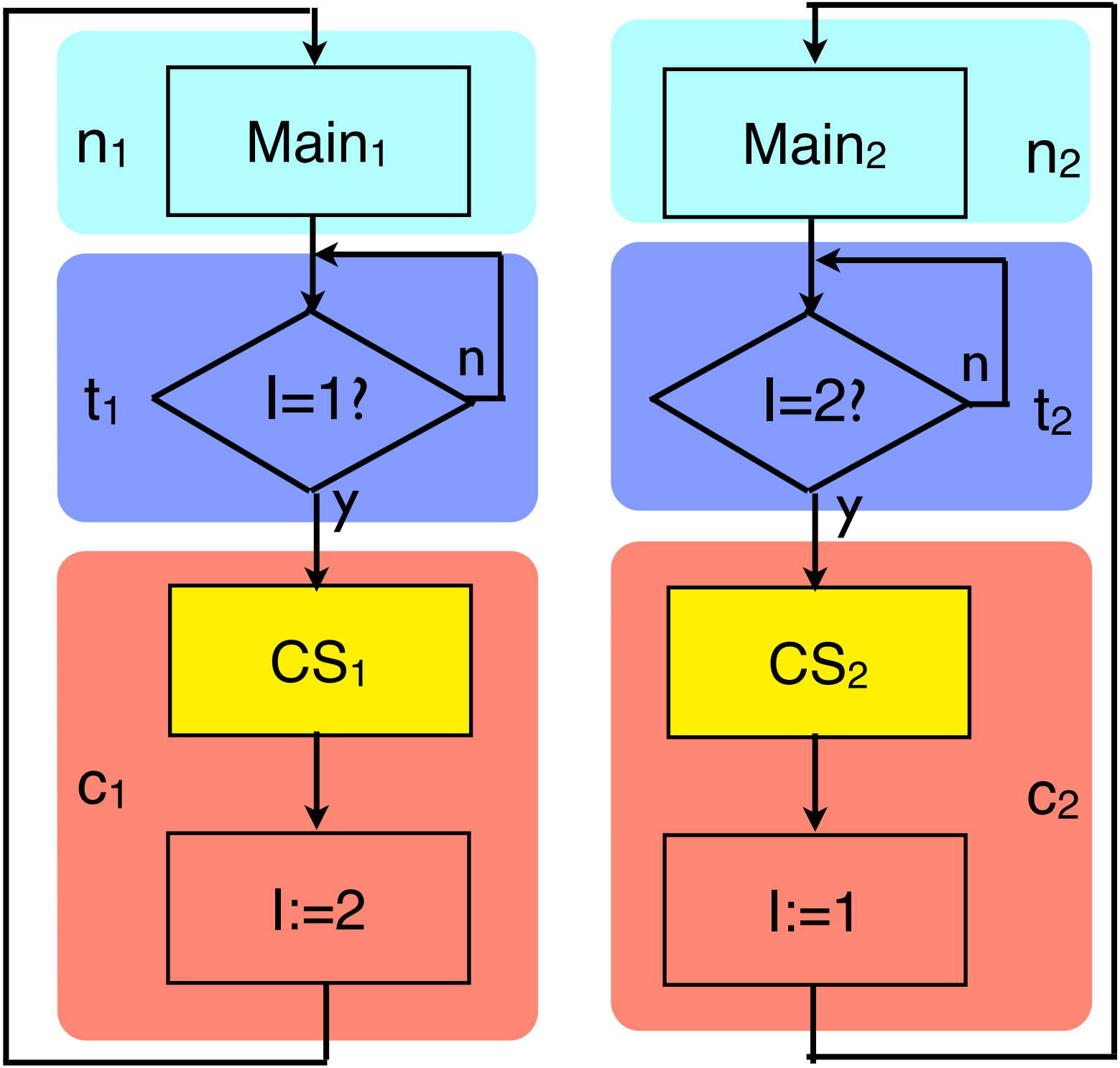

s0
n1n2
s1
t1n2
s5
n1t2
s2
c1n2
s3
t1t2
s9
t1t2
s6
n1c2
s4
c1t2
s7
t1c2

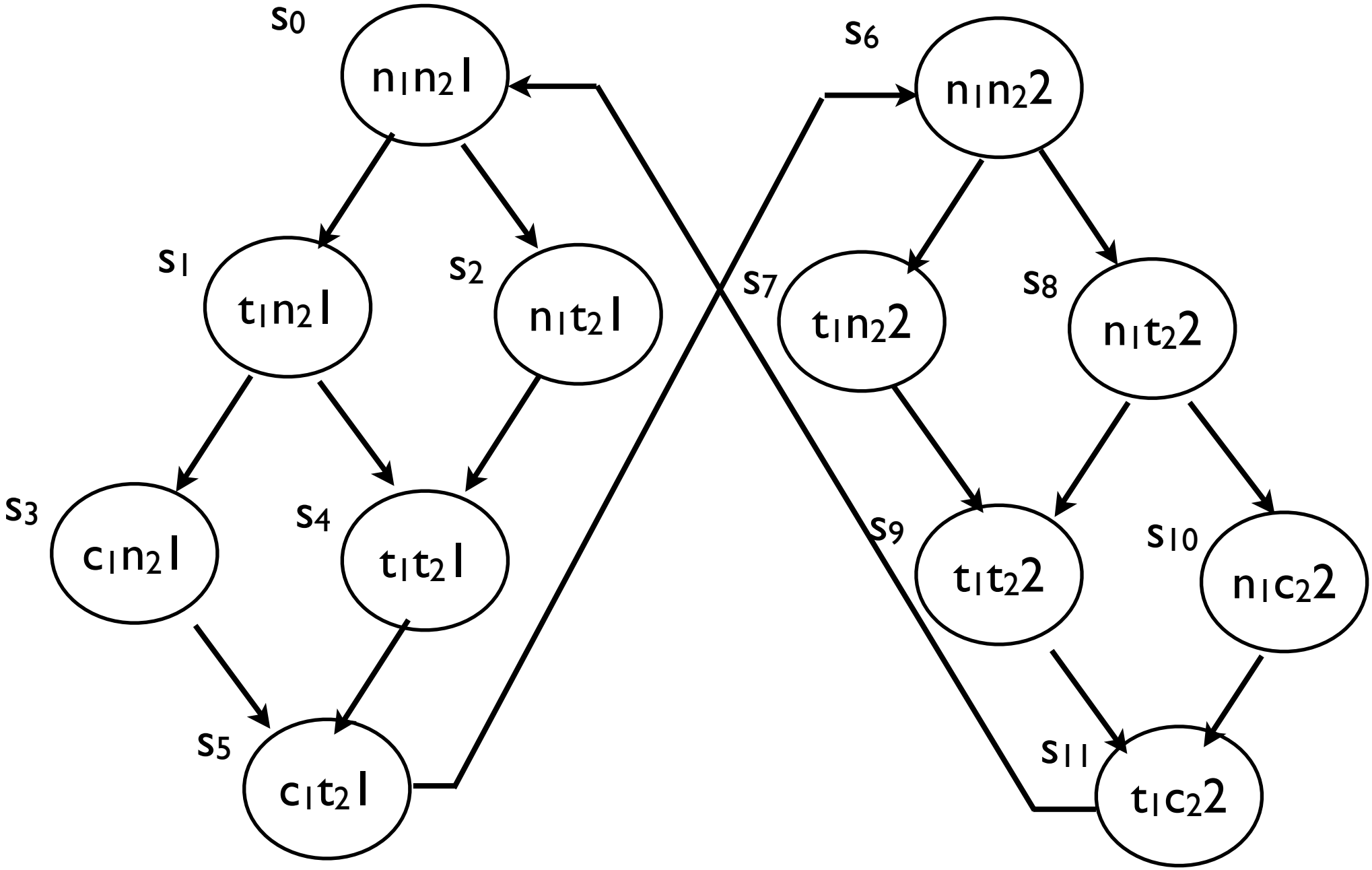
s_3, s_9で，どちらのプロセスが
先にリクエストしたかを記憶する

- Safety
- Liveness
- No strict sequencing

例3

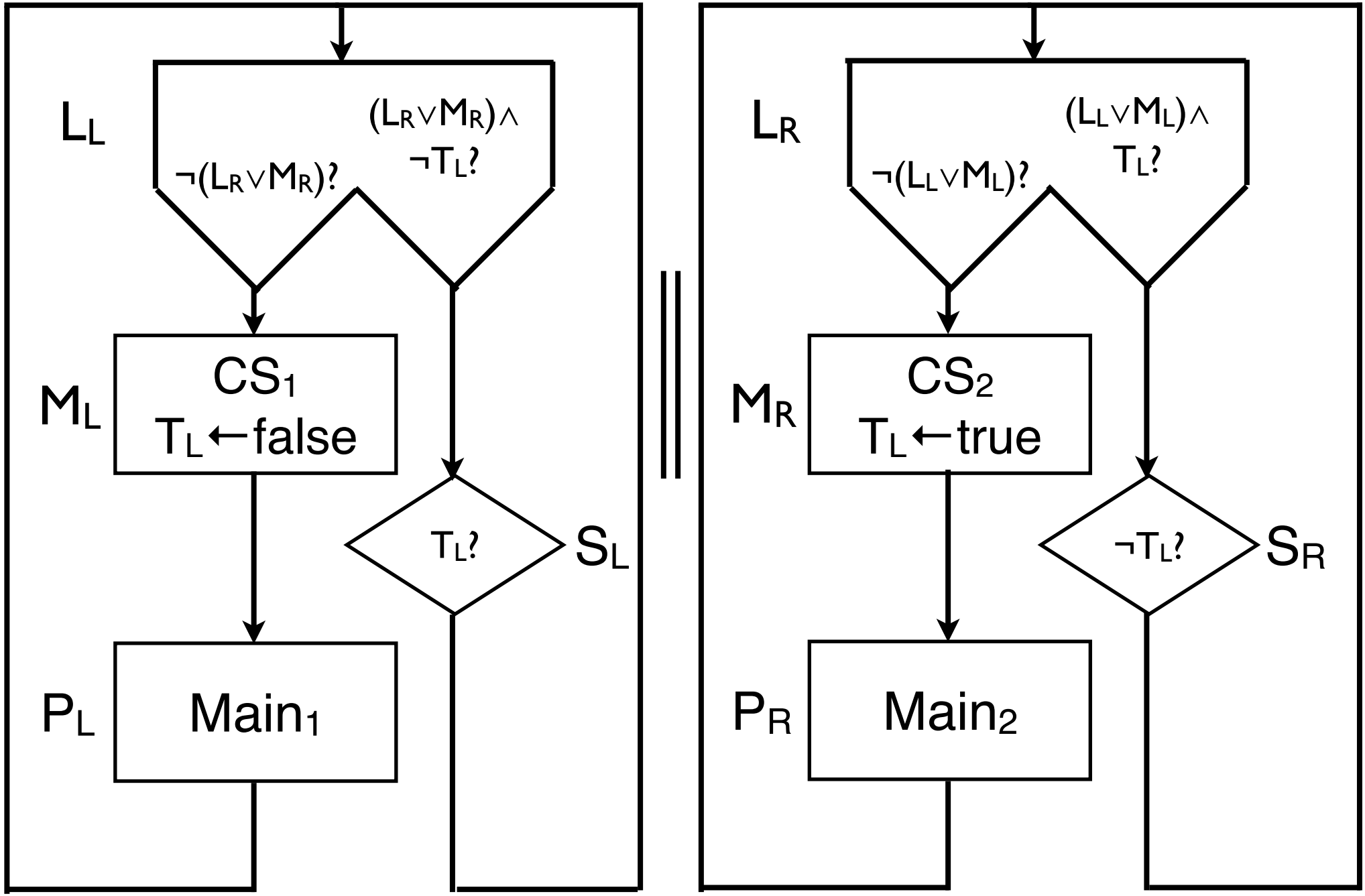
フローチャートによる
「実装」




s0
n1n2 1
s1
t1n2 1
s2
n1t2 1
s3
c1n2 1
s4
t1t2 1
s5
c1t2 1
s6
n1n2 2
s7
t1n2 2
s8
n1t2 2
s9
t1t2 2
s10
n1c2 2
s11
t1c2 2

○ Safety
○ Liveness
(△ Starvation-free: GF c1
相手方のMainが停止する保証がない)
× No strict sequencing

例4 Dekker's Solution (logical version)



- Safety
- Liveness (Mainが停止してもOK)
- No strict sequencing