知能ソフトウェア特論
Intelligent Software

# モデル検査（1）

## 並行システムのモデリング

## Model Checking (1)
## Modeling Concurrent Systems

1. 並行システム
2. モデル検査の概要
3. プロセス代数に基づく逐次プロセスのモデリング
4. 並行プロセスのモデリング

1. Concurrent systems
2. Overview of model checking
3. Modeling sequential processes based on process algebra
4. Modeling concurrent processes

■Reference
Model Checking, E.M. Clarke, Jr. et al, MIT Press (1999)

# 1. 並行システム (Concurrent systems)

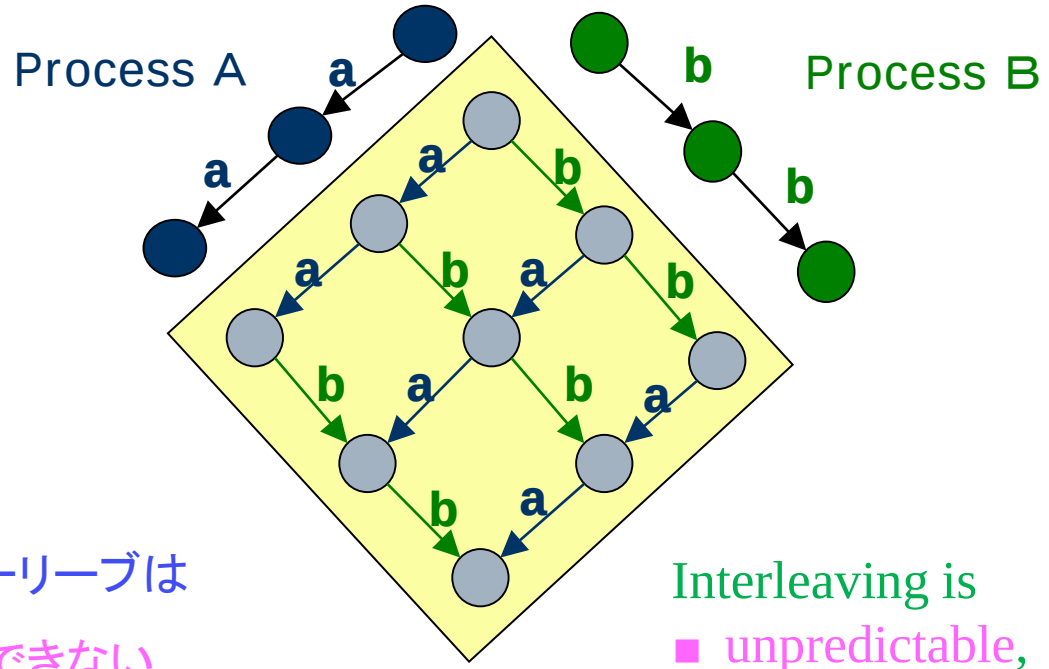
並行システム：
　複数のプロセス（逐次プログラムの実行）が並列（または擬似並列）に動作する計算システム

A concurrent system is a computational system in which multiple processes (i.e., the executions of sequential programs) are run in parallel (or quasi-parallel).

リアクティブ・システム：
　環境からの入力に実時間的に応答する並行システム

A reactive system is a concurrent system that responds to inputs from its environment in real time.

# インタリーブ　(Interleaving)




インターリーブは

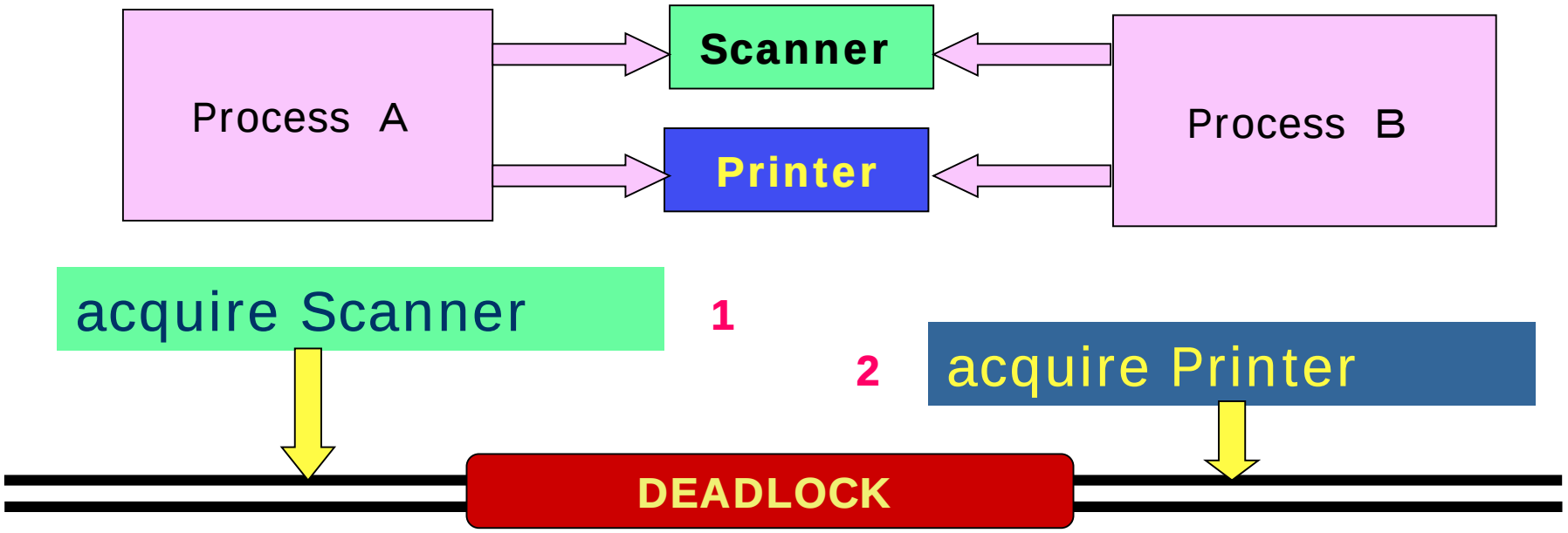
- 予期できない
- 制御できない
- 膨大な数の実行経路を生じる

Interleaving is

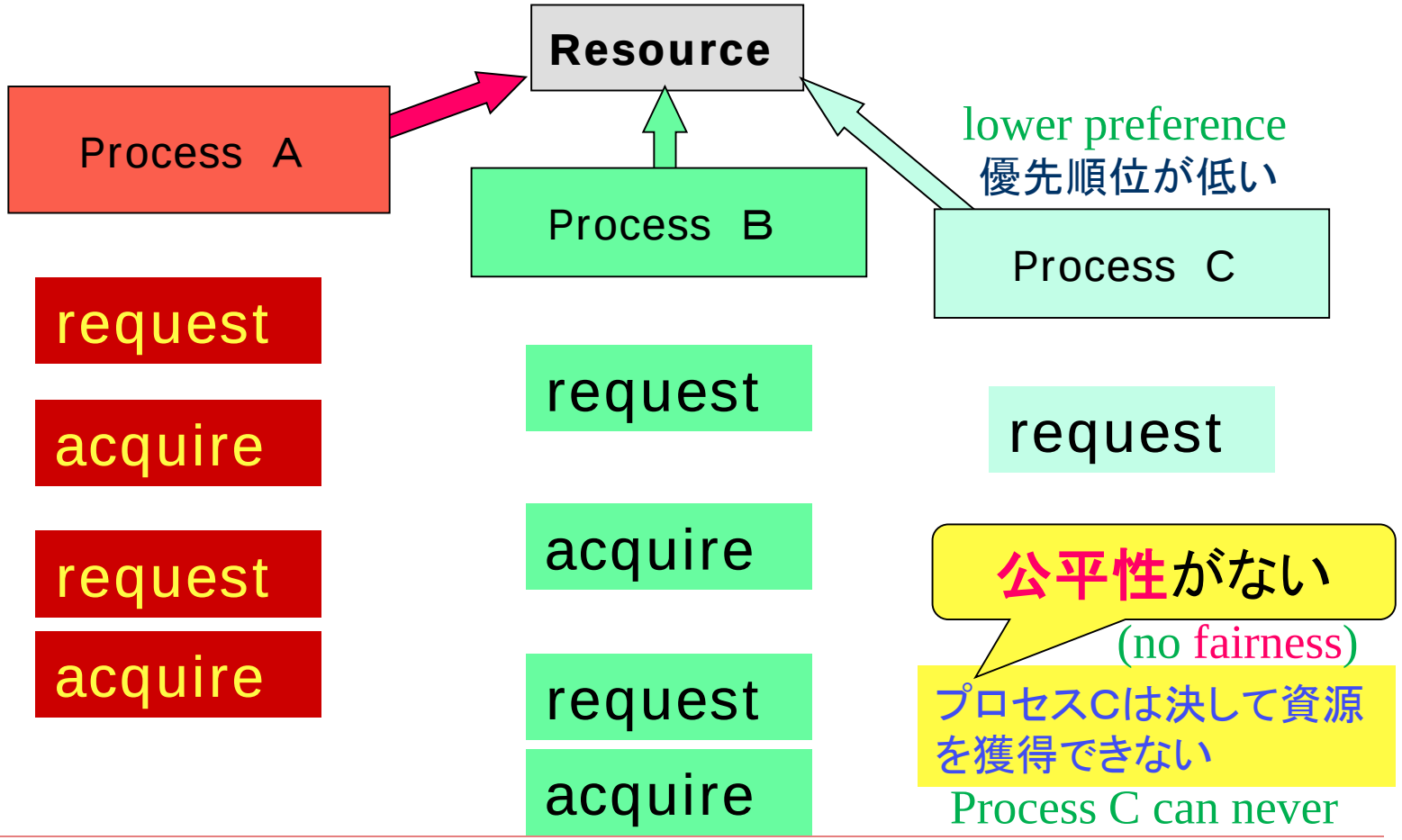
- unpredictable,
- uncontrollable, and
- subject to yielding

a huge number of computational paths.

# デッドロック　(Deadlock)



# ライブロック，飢餓，公平性 (Livelock, Starvation, Fairness)



# 2. モデル検査の概要 (Overview of model checking)

入出力関係に着目した「停止性＋部分正当性」の解析のみでは並行システムの正当性を言えない
→振る舞い（途中の状態遷移）の考慮の必要性

Analysis of input-output relationships such as termination plus partial correctness is not enough to prove the correctness of concurrent systems.
→ Behaviors (i.e., transitions of intermediate states) need to be considered.

モデル検査:
有限状態並行システム の振る舞いの検証を自動で行う技術

Model checking is a technology that automatically verifies the behaviors of finite-state concurrent systems.

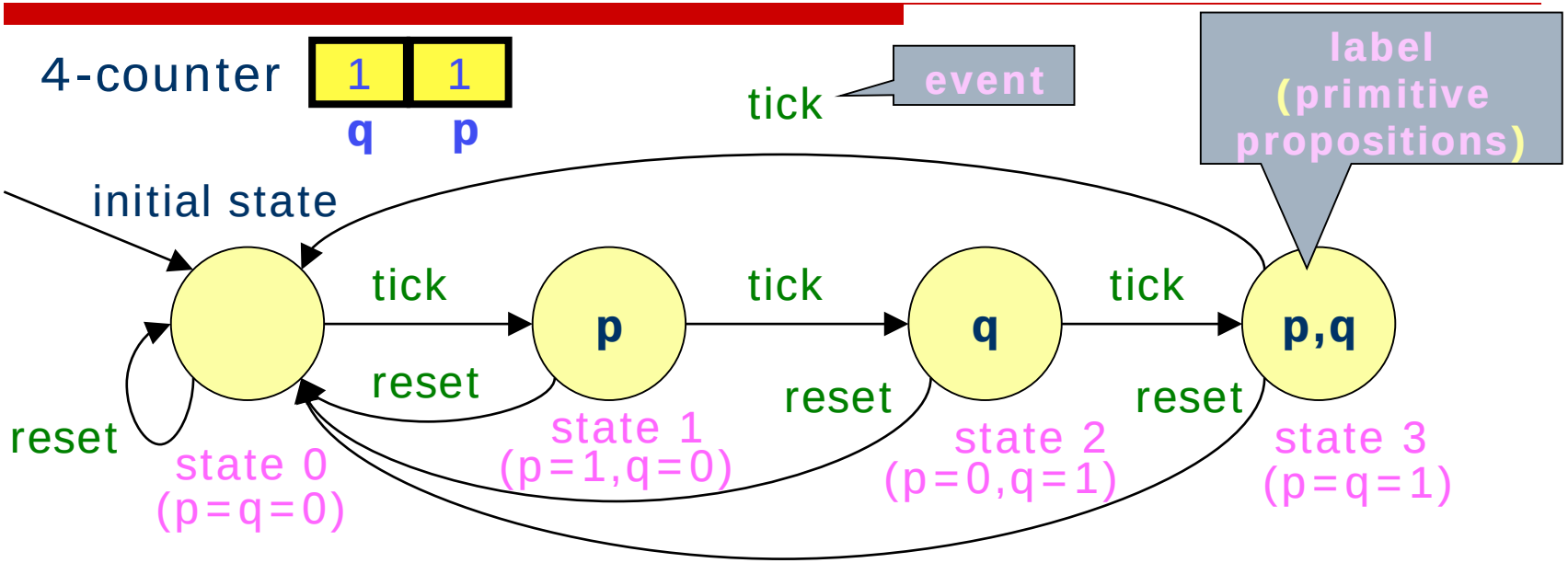
有限状態システム:
状態数が有限個の状態遷移系

A finite-state system is a state transition system with a finite number of states.

検証:
期待される性質(仕様)を満たすことの確認

Verification means to confirm that the system will satisfy the required properties (or specifications).

# 状態遷移系（オートマトン） (State transition system; automaton)




状態遷移系は次の4項目からなる.

- 状態（ノード）の有限集合
- 初期状態の集合
- 遷移関係（有向辺の集合）
- 各状態ごとのラベル

A state transition system consists of

- a finite set of states (nodes),
- a set of initial states,
- a transition relation (a set of directed edges), and
- labels for each state.

状態数は数百万くらいはOK

Millions of states are OK in practice.

# 検証できる主な性質　(Major verifiable properties)

モデル検査では主につぎの2つの性質を検証する

- 安全性 (safety)

「悪いことは決して起こらない」

例：このエレベータは，ドアが開いたまま昇降することは決してない

- 活性(liveness)

「良いことはいつかは成り立つ」

例：資源を要求したら，いつか必ず取得できる

なにが「悪いこと」で，なにが「良いこと」かは，応用目的にあわせて設計者が記述する

Basically, the following two properties can be verified by model checking.

- safety, which ensures that the bad thing will never happen.

Example: This elevator will never start to move while its door is still open.

- liveness, which ensures that the good thing will eventually happen.

Example: If you request a resource, you will eventually acquire it.

What are meant by “bad” or “good” things are described by the system designers, depending on the application.

# モデル検査の実施手順　(Model checking processes)

モデル検査はつぎの3ステップで実施する　Model checking is conducted in the following three processes.

(1) モデリング　(modeling)

(2) 性質の記述　(description of properties)

(3) 検査　(verification)

# (1) モデリング (Modeling)

- システムの振る舞いを表すモデル（状態遷移系）を記述する.
- 記述には一般に，モデル記述言語を用いる.
- プロセス代数に基づく言語やC言語風の言語などがある.

- Describe a model (i.e., state transition system) that defines the behavior of the system.
- Model description languages are used for the description.
- Languages based on process algebra and C-like ones are available.

# (2) 性質の記述 (Description of properties)

- システムが満たすべき性質（プロパティ）を記述する.
- 記述には一般的に，時間の概念を扱う時相論理を用いる.
- 時相論理として，LTLとCTLがよく知られている.

- Describe properties the system is expected to satisfy.
- Temporal logics, which can represent the notion of time, are used for the description.
- Two well-known temporal logics are LTL and CTL.

LTL (Linear Temporal Logic)　　CTL (Computation Tree Logic)

【LTL Example】 **G**(Req → **F** Ack)


Reqが真になると，その後いつか必ずAckが真になる

Whenever Req is ON, then Ack will eventually become ON in the future.

It is **globally (always)** true that
if Req is true then in **future** Ack will be true.

# (3) 検査 (Checking)

モデルが，記述された性質を満たすかどうか，検査する.

Check to see if the model satisfies the described properties.

モデル検査器（model checker）

A model checker is used for checking.

- 性質が成り立つ場合：検査終了
- 性質が成り立たない場合：
  反例（エラートレース）を出力

- if the properties hold, the check is finished.
- Otherwise, a counterexample (error trace) is output.

モデル検査アルゴリズムの原理

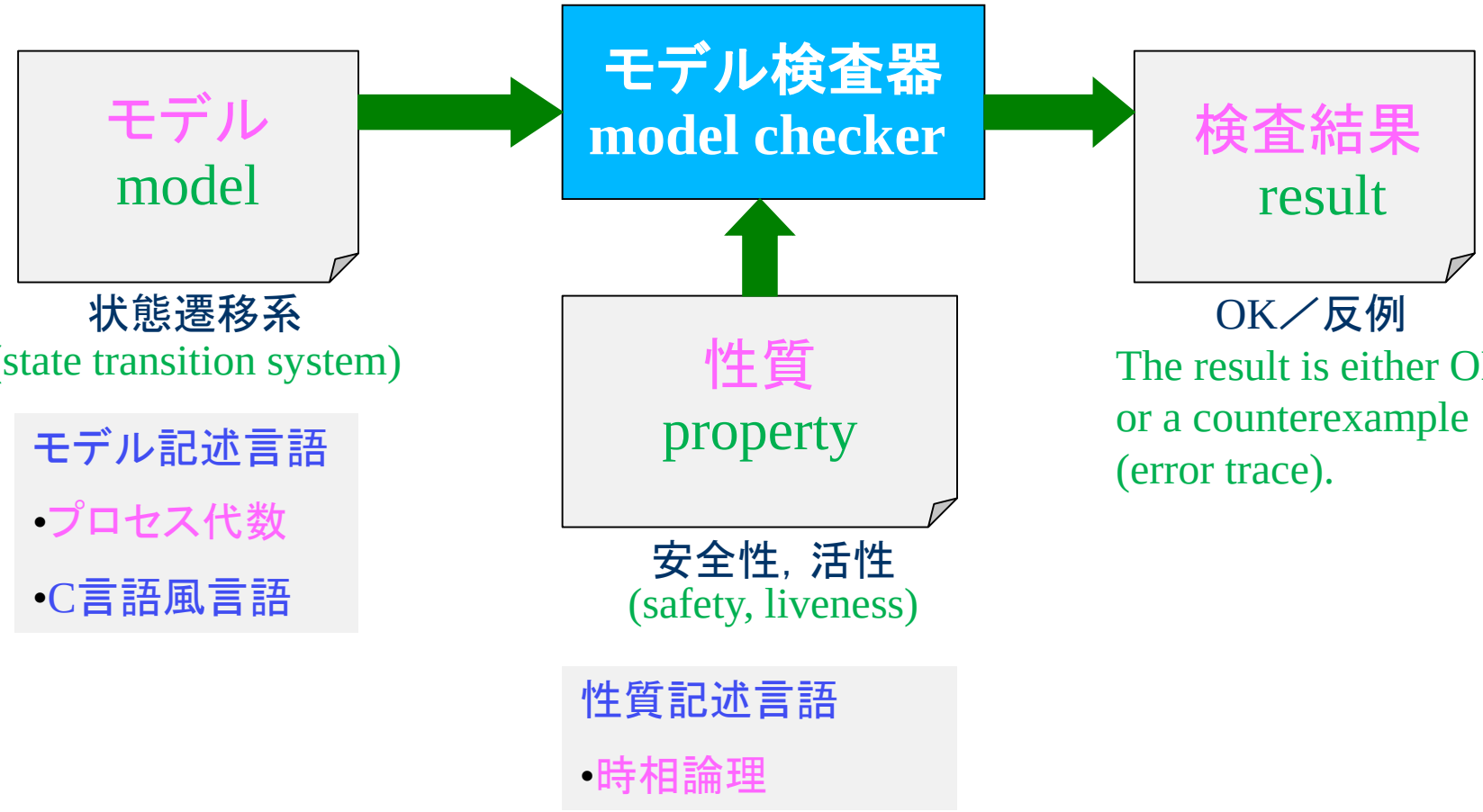
すべての入力について計算経路を網羅的に検査

The principle of model checking algorithms is to examine all computation paths for all possible input.

# モデル検査器の例　(Well-known model checkers)

- SMV, NuSMV
  - カーネギーメロン大学　(Carnegie Mellon University)
  - IEEE Futurebus+ standardのバグ発見　(Found bugs of an IEEE bus)
- SPIN
  - ベル研究所　(Bell Laboratories)
  - ACM Software System Award受賞　(Received the ACM SS Award)
- LTSA
  - ロンドン王立大学　(Imperial College London)
  - FSP言語，アニメーション　(FSP Language, amination)
- Java PathFinder (JPF)
  - NASA　(National Aero. & Space Admin.)
  - Javaのバイトコードの検査　(Checks Java byte code)

# モデル検査器の概要　(Overview of model checker)

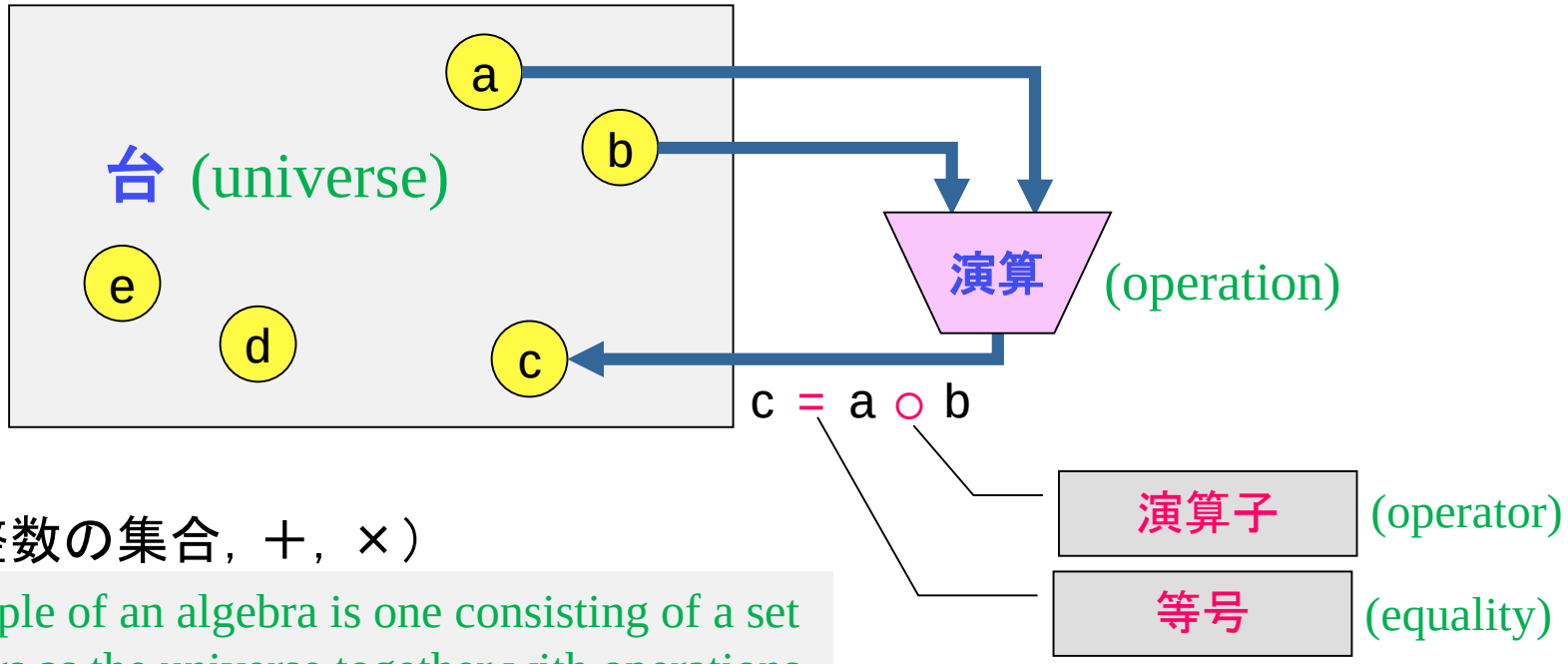


# 3. プロセス代数に基づく逐次プロセスのモデリング
## (Modeling sequential processes based on process algebra)

代数とは:

要素の集合(台)およびその上での演算の集まりからなる計算系

An algebra is a computational system consisting of a set of elements (called the universe) and a collection of operations on those elements.



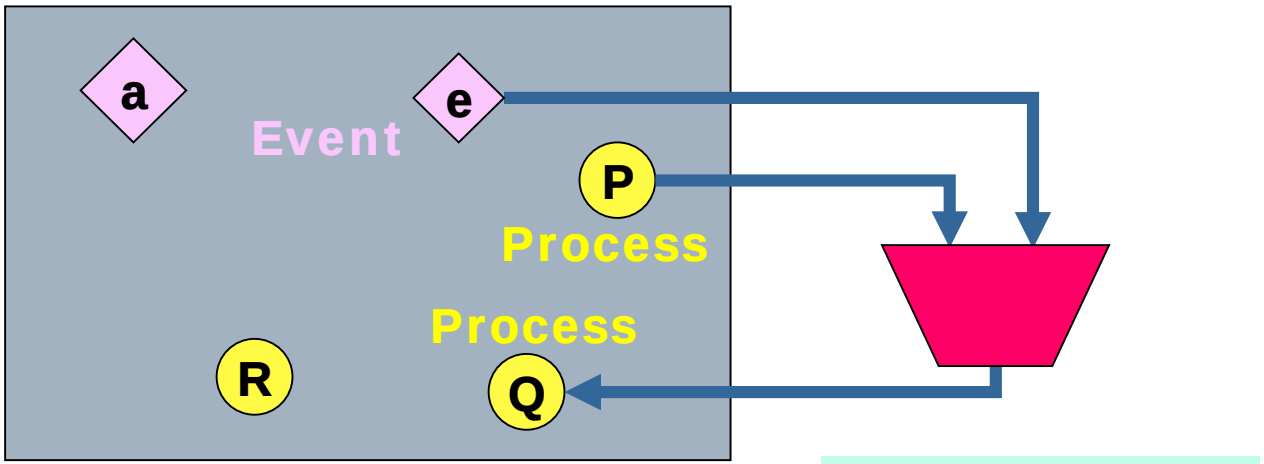

例: (整数の集合, +, ×)

An example of an algebra is one consisting of a set of integers as the universe together with operations + and ×.

# プロセス代数とは (What is process algebra ?)

プロセスやイベント等の集合を台とし，
プロセスの合成などの演算を定義した代数

A process algebra is an algebra consisting of processes and events as the universe together with operations such as process compositions.




$Q = e \circ P$

# アクション，状態遷移，プロセス (Action, State transition, Processes)

- 状態：プログラムカウンタの値と（明示的あるいは暗黙的な）変数の値からなる

  A state consists of the value of the program counter and other (explicit or implicit) variables.

- 状態遷移：割り込み不能の命令などを表すアクションにより状態が遷移する

  A state transition occurs by atomic actions such as execution of uninterruptible instructions.

- プロセス：実行中の1つの逐次プログラムのこと．その時点以降で可能なアクションの列（トレース）の集合によってモデル化される

  A process represents a sequential program under execution. It is modeled by a set of traces, which are sequences of actions possibly seen in the future.

# ラベル付き遷移系とFSPによるモデリング

## (Modeling by labeled transition systems and FSP)

モデルは，ラベル付き遷移系として定義され，FSPなどのモデル記述言語によって記述される.

A model is defined as a labeled transition system which can be described in a model-description language such as FSP.

- ラベル付き遷移系：有限状態機械を図式的に表現したシステム

  A labeled transition system represents a finite-state machine in a graphical manner.

- FSP：ラベル付き遷移系を代数的に表現する言語

  FSP (Finite State Processes) represents labeled transition systems in an algebraic manner.

# アクション接頭辞　(Action prefix)

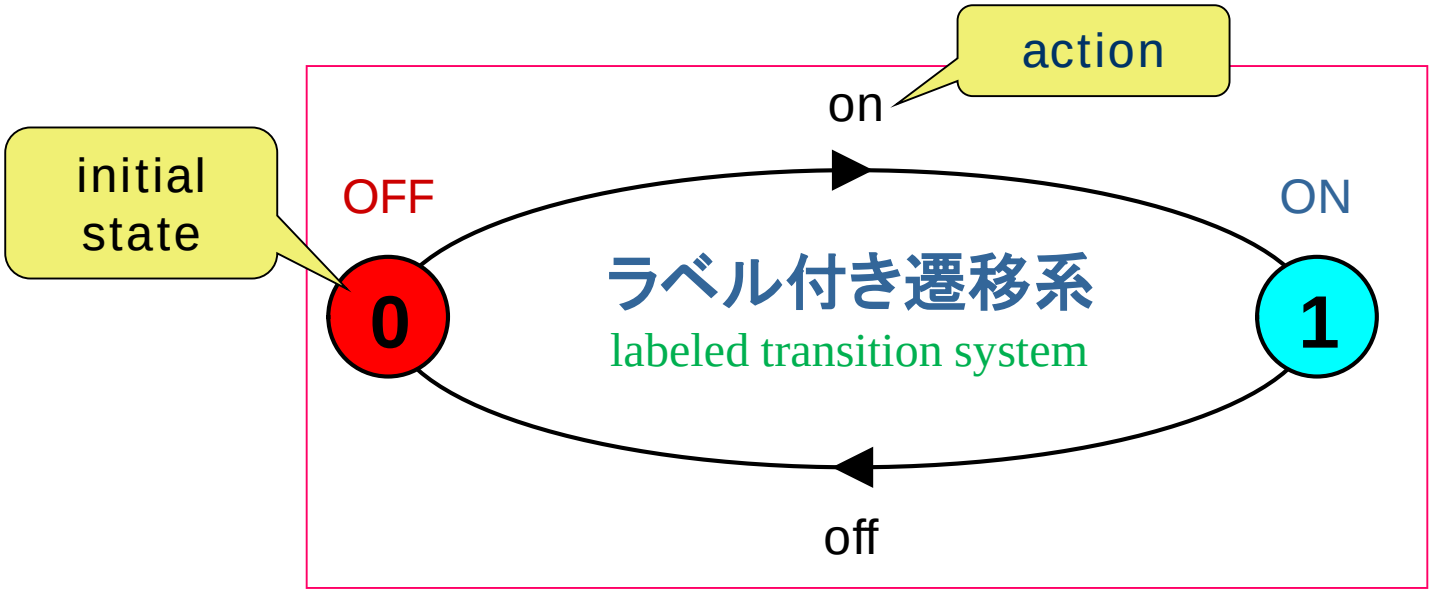
**(a-> P)**

最初にアクション **a** を実行し，つぎにプロセス **P**と同じ振る舞いをするプロセス.

(a -> P) is a process that initially engages in the action **a** and then behaves exactly as described by the process **P**.

# スイッチの例(1) ラベル付き遷移系のモデル

(Example of a binary switch: model as labeled transition system)




トレース：実行可能なアクションの列のこと

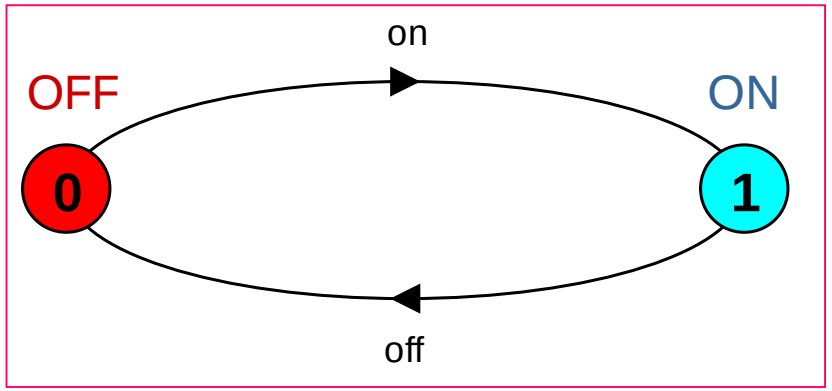
Trace: a possible action sequence

**on→off→on→off→on→off→ ……**

# スイッチの例(2) FSPによる記述
## (Example of a binary switch: description in FSP)

FSP

```
SWITCH = OFF,
OFF    = (on -> ON),
ON     = (off-> OFF).
```




Process name in capitals

Action name in lower-case

代入によってより簡潔な表現を得る (Simplify the expressions by substitution)

```
SWITCH = OFF,
OFF    = (on ->(off->OFF)).
```

```
SWITCH = (on->off->SWITCH).
```

# 選択 (Choice)

**(a-> P | b->Q)**

最初にアクション **a** ,**b** のいずれかを実行する.
そのアクションが **a** ならつぎにプロセス **P** を実行し,
**b** ならつぎにプロセス **Q** を実行するプロセス.

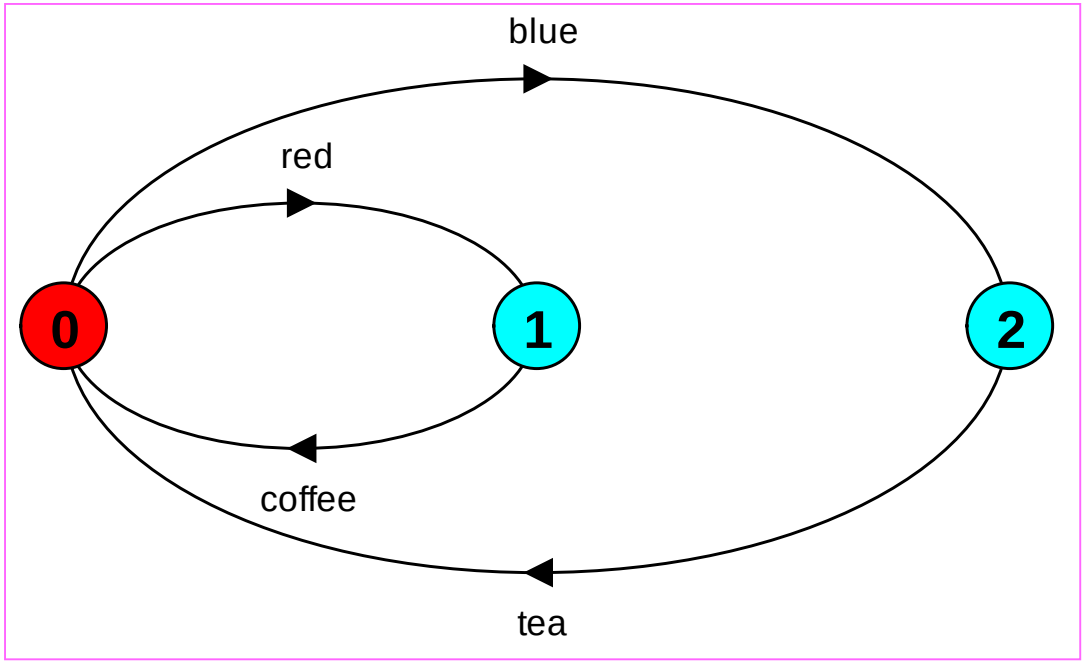
(a->P | b->Q) is a process which initially engages in either a or b. The subsequent behavior is described by P if the first action was a, and Q if the first action was b.

# 自動販売機の例 (Example: a vending machine)

```
DRINKS = ( red->coffee->DRINKS
         | blue->tea->DRINKS
         ).
```

ボタンの色で飲み物を指定

Select a drink by the color of the button



# 4. 並行プロセスのモデリング (Modeling concurrent processes)

## プロセスの並列合成 (parallel composition of processes)

(P||Q)

プロセス P と Q の並行実行を表すプロセス.
|| は並列合成演算子.

This expression means the process in which processes P and Q are executed in parallel.
The symbol || denotes the parallel composition operator.

**Commutative:** (P||Q) = (Q||P)
**Associative:** (P||(Q||R)) = ((P||Q)||R)
= (P||Q||R).

# アクションのインタリーブ　(Action interleaving)

かゆいところをかくプロセス

a process that itches

プロセス間で共有されないアクションはインタリーブされる

Actions not shared among the processes will be interleaved.

```
ITCH  = (scratch->STOP).
CONVERSE = (think->talk->STOP).

||CONVERSE_ITCH = (ITCH || CONVERSE).
```

共有アクション無し
no shared actions

会話するプロセス

a process that converses

並列合成は || で書き始める

並列合成されたプロセス

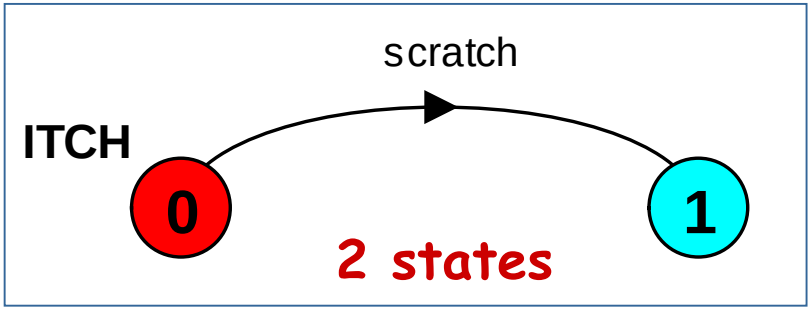
the process created by parallel composition

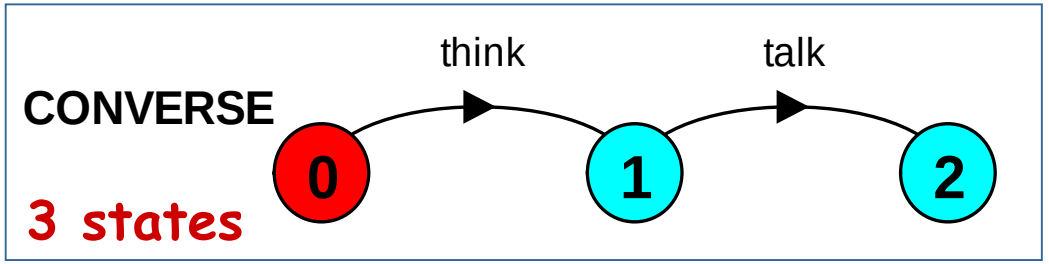
In FSP, a parallel composition statement starts with ||.

```
think→talk→scratch
think→scratch→talk
scratch→think→talk
```

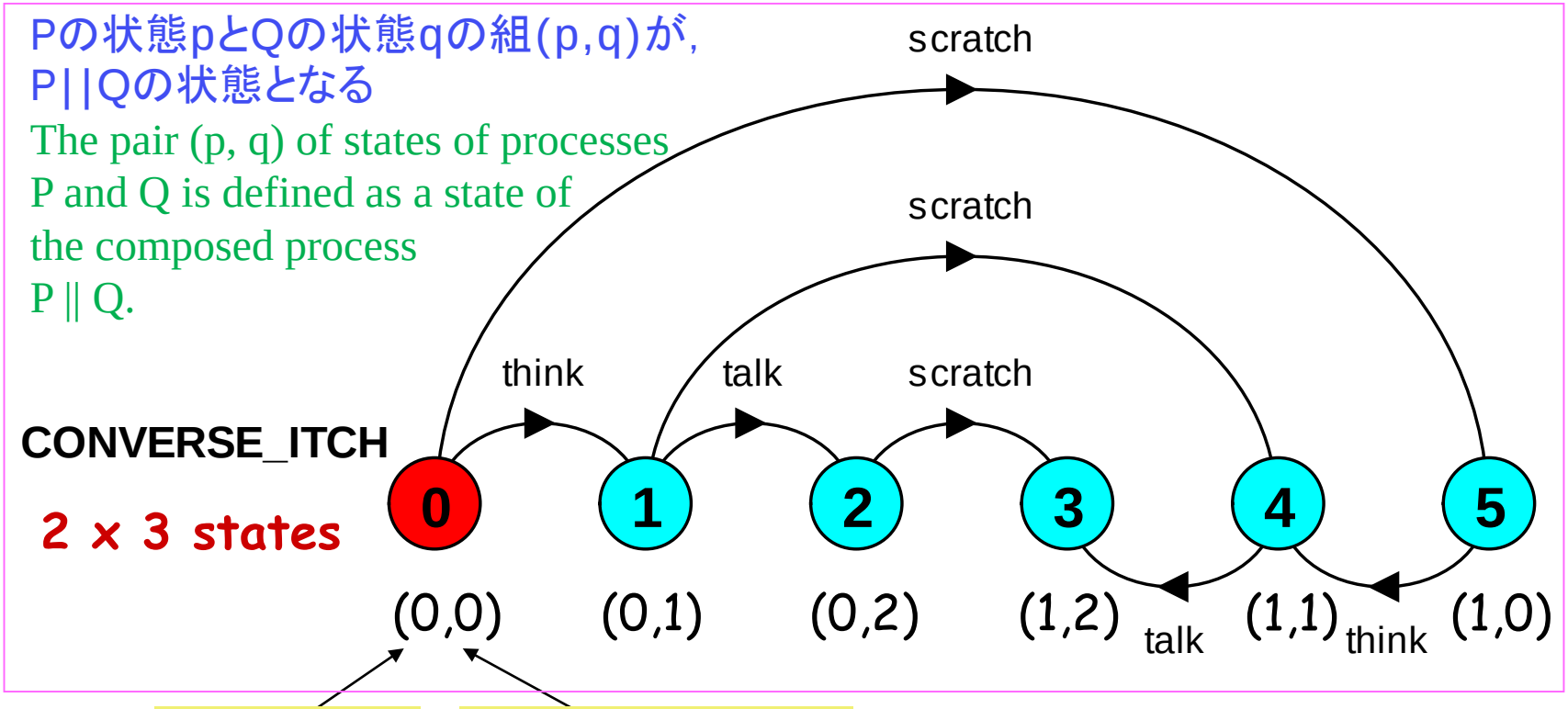
インタリーブによって可能となるトレース
the traces made possible by interleaving

# ラベル付き遷移系の並列合成
## (Parallel composition of labeled transition systems)




Pの状態pとQの状態qの組(p,q)が,
P||Qの状態となる
The pair (p, q) of states of processes P and Q is defined as a state of the composed process P || Q.



# 共有アクションとインタラクション
## (Shared action and interaction)

**共有アクション**：**並列合成されたプロセスがもつ共通のアクション.**

**共有アクションによってプロセスの**インタラクション**をモデル化.**

**共有アクションは，それを共有するすべてのプロセスにおいて同時に同期して実行されなければならない.**

A shared action is an action shared among the parallel processes.
Shared actions are used to model interactions among those processes.
Shared actions must be executed at the same time by all processes that share them.
In other words, those executions must be synchronized.

# インタラクションの例

(Example of interactions)

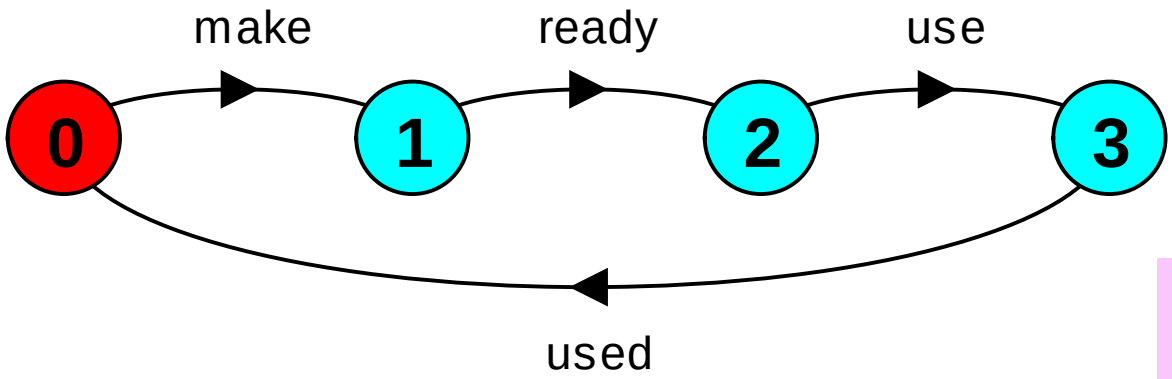
shared actions:
ready, used

```
MAKER = (make->ready->used->MAKER).
USER  = (ready->use->used ->USER).

||MAKER_USER = (MAKER || USER).
```

3 状態
3 状態

3 x 3 状態?




4 状態

インタラクションは
振舞いを制約する

Interactions constrain the behavior.

# 演習問題 3
## EXERCISE 3

二人のユーザーを表すプロセスUSER_A, USER_B, および共有資源を表すプロセスRESOURCE, およびそれらを並列合成したプロセスRESOURCE_SHAREが, FSPでつぎのように定義されている. これら4つのプロセスのラベル付き遷移系をそれぞれ描画せよ.

The processes USER_A and USER_B, which represent two users, and RESOURCE , which represents a shared resource, are combined by the pallarel composition to define RESOURCE_SHARE in FSP. Draw the labeled transition systems for these four processes.

```
USER_A = (a_acquire -> a_use -> a_release -> USER_A).

USER_B = (b_acquire -> b_use -> b_release -> USER_B).

RESOURCE = IDLE,
   IDLE  = (a_acquire -> BUSY  |  b_acquire -> BUSY),
   BUSY = (a_release -> IDLE   |  b_release -> IDLE).

||RESOURCE_SHARE = (USER_A || USER_B || RESOURCE).
```