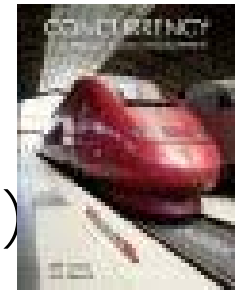
経営情報システム学特論１
２．並行システムモデリングの基礎

SS専攻 経営情報システム学講座 客員
石川 冬樹
f-ishikawa@nii.ac.jp

進め方（再）

- マルチスレッドプログラミングで体験しつつ，モデルの記述・検証も行っていく
  - 分散システム（コンピューター複数）の場合は，プログラミング演習まではしない
- 教科書（買わなくてもよい）
  - Concurrency: State Models and Java Programs, Jeff Magee and Jeff Kramer, Wiley 2006



- 演習
  - Java（講師はEclipseにて説明，他でもよい）
  - LTSA

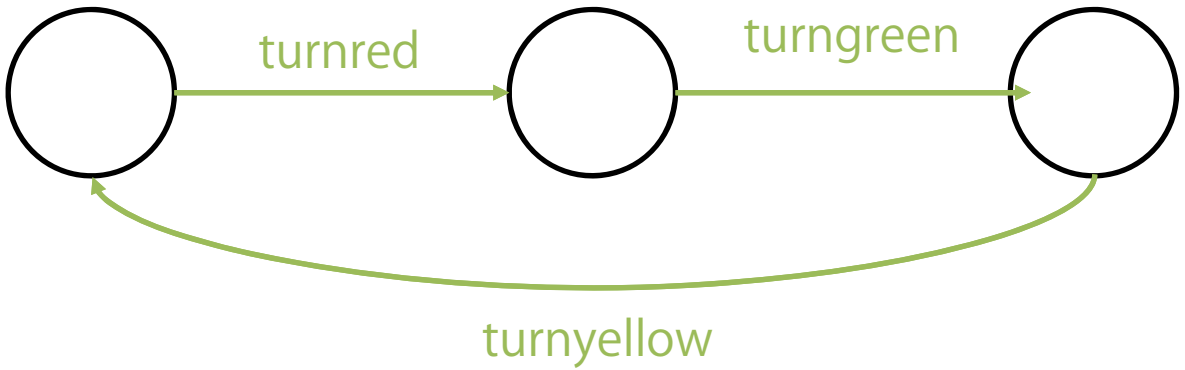
http://www.doc.ic.ac.uk/~jnm/book/

目次

- Labeled Transition Systemの基礎
- Javaマルチスレッドプログラミングの第一歩
- 簡単な例題

Labelled Transition System（LTS）

- State Machine（状態機械）
  - ノードは状態を表し，遷移・アクションによってつながれる
  - cf. フローチャートではノードがアクション
  - 遷移・アクションがラベルにて識別されるもの
- *Labelled Transition System*



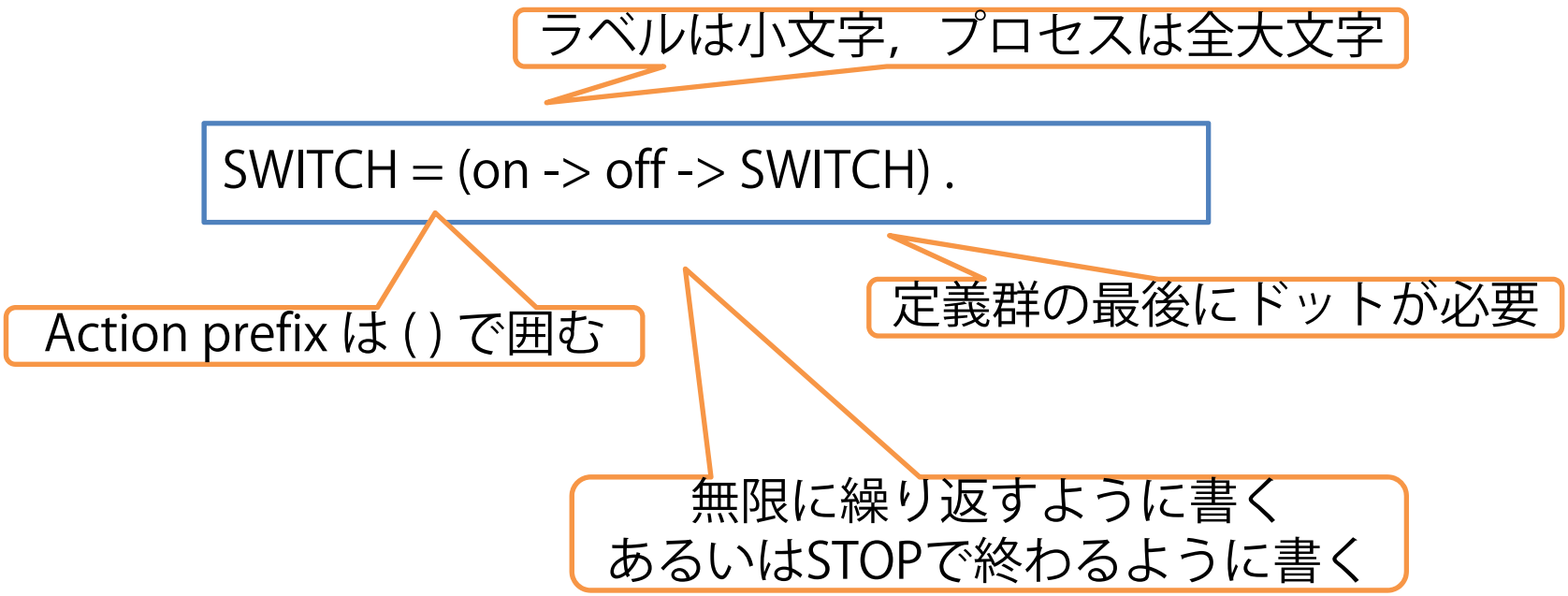
Finite State Process

- Process algebra（プロセス代数）は，並行プロセスやそれにより構成されるLTSの記述によく使われる
  - CSP（Communicating Sequential Process），π-Calculusなど
  - 「代数的」：文字を記号として用いて・・・
  - 使用例：設計記述の本質をこれらの記述として抜き出し検証を行う
  - 使用例：これらの記述から並行処理プログラムを生成する

本講義ではLTSAツールが読み込むFinite State Processes（FSP）というものを用いる

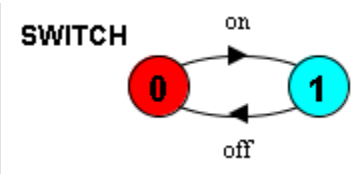
Action Prefix

- アクションx，プロセスPに対して，
  （x -> P）
  は，xを行った後にPとして振る舞うことを表す



LTSAの使い方（１）

- Editタブで入力
- BuildメニューからCompile
- Drawタブにて選択すると状態遷移を図示



Process Definition

- 下記は同じ

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

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

内部的に用いる複数の
サブプロセス定義の区切りはコンマ

Choice

- アクションx，y，プロセスP，Qに対して，

  （ x -> P | y -> Q ）

  は，xを行った後にPとして振る舞うか，yを行った後にQとして振る舞うかのいずれか片方を（非決定的に）行うことを表す

```
VENDING = ( button1 -> coffee -> VENDING
          | button2 -> tea -> VENDING).
```

Index and Parameters

- 下記は同じ

```
BUFF = ( in[0] -> out[0] -> BUFF
       | in[1] -> out[1] -> BUFF
       | in[2] -> out[2] -> BUFF ).
```

```
BUFF = ( in[i:0..2] -> out[i] -> BUFF ).
```

```
BUFF(N=2) = ( in[i:0..N] -> out[i] -> BUFF ).
```

```
range T = 0..2
BUFF = ( in[i:T] -> out[i] -> BUFF ).
```

```
const N = 2
range T = 0..N
BUFF = ( in[i:T] -> out[i] -> BUFF ).
```

Guarded Action

- アクションx，y，プロセスP，Q，ガード条件Bに対して，

  (when B x -> P | y -> Q)

  は，

  Bが成り立つ場合 x -> P も y -> Q のいずれかとして振る舞う可能性があり，

  そうでない場合， y -> Q として振る舞う

  ことを表す

```
ELEVATOR (N=3) = ELEVATOR[0],
ELEVATOR[i:0..N] = ( when (i<N) up -> ELEVATOR[i+1]
                   | when (i>0) down -> ELEVATOR[i-1] ).
```

Parallel Composition

- プロセスP，Qに対して，
  （P || Q）
  は，PとQの並行実行（合成）を表す

```
WORKER_A = ( work_a1 -> work_a2 -> STOP ).
WORKER_B = ( work_b1 -> work_b2 -> STOP ).
|| WORKERS = ( WORKER_A || WORKER_B ).
```

合成プロセスの場合名前の最初に || を付ける

LTSAの使い方（２）

- 合成プロセスの場合，Compileの後にCompose を

Shared Actions

- 複数プロセス内で同じラベルがあった場合，同期される

```
CURRYRICE = ( order -> curryrice -> CURRYRICE ).
SALAD = ( order -> salad -> SALAD ).
||SERVER = ( CURRYRICE || SALAD ).

GUEST = ( order -> GUEST ).

||RESTAURANT = ( GUEST || SERVER ).
```

Label on Multiple Process Instances

- 同じ種類のプロセスを複数作りたい場合に，プロセスにprefixを付けると，その中のラベルすべてにprefixを付けることができる

```
CURRYRICE = ( order -> curryrice -> CURRYRICE ).
SALAD = ( order -> salad -> SALAD ).
||SERVER = ( CURRYRICE || SALAD ).

GUEST = ( order -> STOP ).

||RESTAURANT =
   ( a:GUEST || b:GUEST || {a,b}::SERVER ).
```

Label on Multiple Process Instances

- （続）

```
P1 = ( share -> finish -> STOP ).
P2 = ( work -> share -> STOP ).

||WORLD = ( a:P1 || {a,b}::P2 ).
```

aかbかのどちらか1つを
付けたプロセス1つ

```
P1 = ( share -> finish -> STOP ).
P2 = ( work -> share -> STOP ).

||WORLD = ( a:P1 || {a,b}:P2 ).
```

aとbをそれぞれ
付けたプロセス2つ

Relabeling (Substitution)

- ラベルの置換が可能
  - 内部ラベル名を外部ラベル名に変えて同期させるなど

```
P1 = ( send -> finish -> STOP ).
P2 = ( receive -> exit -> STOP ).

||WORLD = ( P1 || P2 ) /{receive/send}.
```

Hiding

- インターフェースとなるラベルを定義し，それ以外のラベルを他プロセスから隠すことができる（内部のラベルはtau・τとされる）

```
P1 = ( work -> finish -> STOP ) @{finish}.
P2 = ( work -> exit -> STOP ).

||WORLD = ( P1 || P2 ).
```

目次

- Labeled Transition Systemの基礎
- Javaマルチスレッドプログラミングの第一歩
- 簡単な例題

Javaでのプログラミング

- スレッドが行う動作を定義するクラス
  - Threadクラスをextends
    あるいは
  - Runnableインターフェースをimplements
  - いずれにしてもrunメソッド内に，動作内容を記述

- 該当クラスのオブジェクトを作成し，
  start/join/sleep/yield メソッドで操作

付録：ThreadTest.java

目次

- Labeled Transition Systemの基礎
- Javaマルチスレッドプログラミングの第一歩
- 簡単な例題

例題： CountDown

- 付録： CountDown.java
  - アプレット（通常ブラウザにて表示されるGUIプログラム，だいぶ古い仕組みだが・・・）
    - ロードされるとinitが実行され，開始・表示されるとstartが実行される（startは何度も発生しうる）
  - Runnableインターフェースも実装しており，自身がスレッドとしての動作も定義している
- 問題：このJavaオブジェクトから立ち上げられるスレッドの挙動を表すFSPを書いてみよ
  - 「画面に表示する」ということはラベルで捨象する
  - 次に，ブラウザ（initとstartを呼び出す側）や時計の動きも加えてみる

まとめ

- プロセス代数による並行プロセスの記述
  - 数式としてのプロセス記述
- オブジェクト指向での典型的なマルチスレッドプログラミング
  - 特定メソッドでの動作定義

- 次回： 相互排他を扱う