しずかにきしれ四輪プログラム
ー状態とデーターー

岐阜経済大学　経営学部 経営情報学科　井戸　伸彦

来歴:

０．０版　２００３年５月２４日
１．０版　２００４年９月２４日:IIOSSからJudeへの切り替え
２．０版　２００５年１１月８日:全般見直し
２．１版　２０２３年６月１５日:(3.2.2)修正

スライドの構成

はじめに
（１）状態とデータ
（２）ツールを用いた状態遷移図作成
（３）状態遷移の例
（４）シーケンス図
（５）分岐とデータ
（６）状態を持つ2つのもの
（７）状態の階層化
（８）データ構造
おわりに

はじめに

■受講者は、初歩のプログラミングの経験があるものとしています。

■ツールとして、Judeを用います。次のサイトからダウンロードできる、フリーソフトウェアです。

- http://objectclub.esm.co.jp/Jude/jude.html

Judeは、UMLのためのツールであり、この講座の目的とは少しずれています。このため、やや変則的な使用方法も行います（UMLについては、別講座にて学びます）。

■説明は直感的な分かりやすさを重視し、厳密には不正確な言い方もしています。

■文中、“オブジェクト指向”との言葉が何回か現れますが、これについては、別講座で取り上げます。

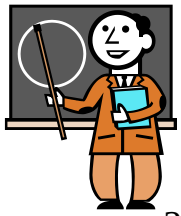
■掲載したプログラムは、いくぶんコーディング規約を破っています。見易さのために、そうしました。

（1．1）まずは、思考実験

■Aさんの生活についての、次の記述を読んでください。

- A君は下宿で寝ている時に目覚まし時計がなると、服を着て登校する。
- 登校中に汗ばめば、帰宅してシャワーに入り眠る。
- 学校に着くと、食堂で食事をして講義に出る。
- 講義中に頭が痛くなると、うがいをしてから散歩に出る。
- 散歩中に猫に会うと、叫び声をあげ、下宿に逃げ帰って寝る。
- 散歩中に学校に着くと、おやつを食べてから、講義に出る。
- 講義が終わると、お茶を飲んでから、図書館で勉強する。
- 図書館で勉強中に友達にあうと、一緒に遊んでから、帰って寝る。
- 図書館で勉強中に図書館が閉まると、食事をして帰って寝る。

■Aさんの生活について、想像がつきましたか？



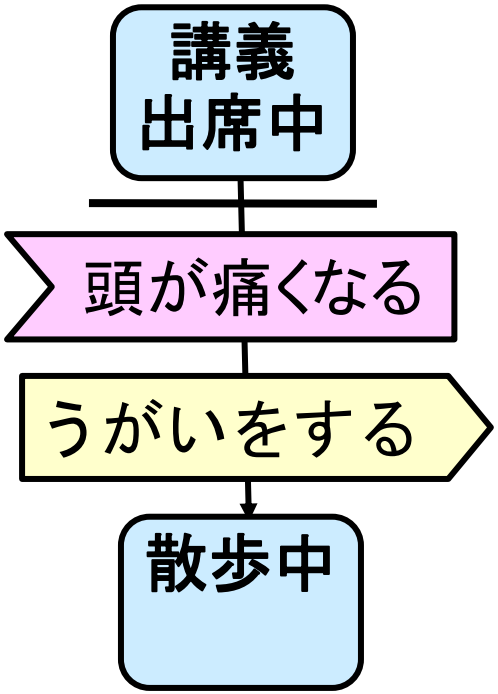
（1．1．1）状態

■Aさんの生活を考える時、状態ということが問題になってきます。

■例えば、「講義中に頭が痛くなると、散歩に出る」という規則は、次のように考えられます。

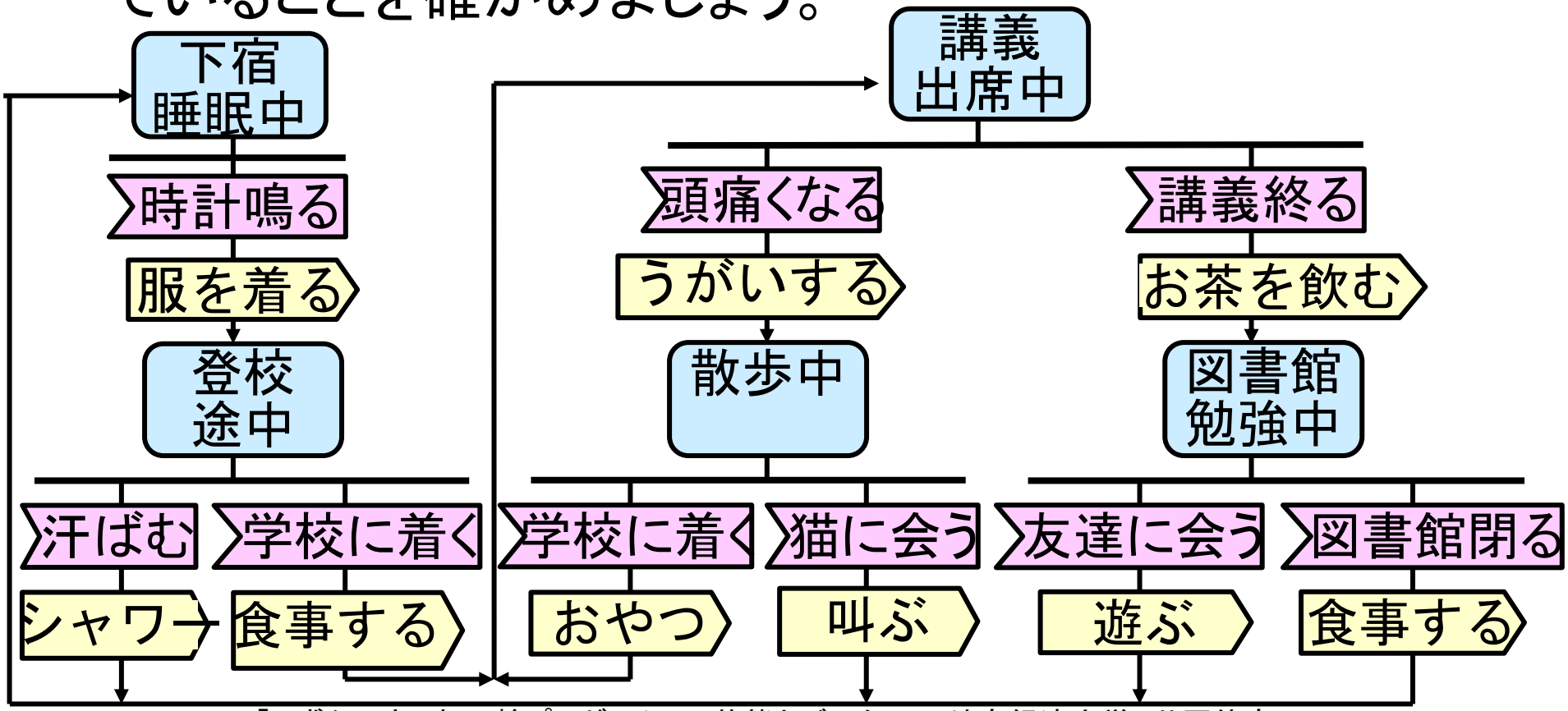
- “講義出席中”という状態がある。
- “散歩中”という状態がある。
- “講義出席中”に、
  “頭が痛くなる”ということが起こると、
  “散歩中”に移る（遷移する）。
  移る際には、うがいを実行する。

■上記のような考え方を、
右図のように表すこととします。



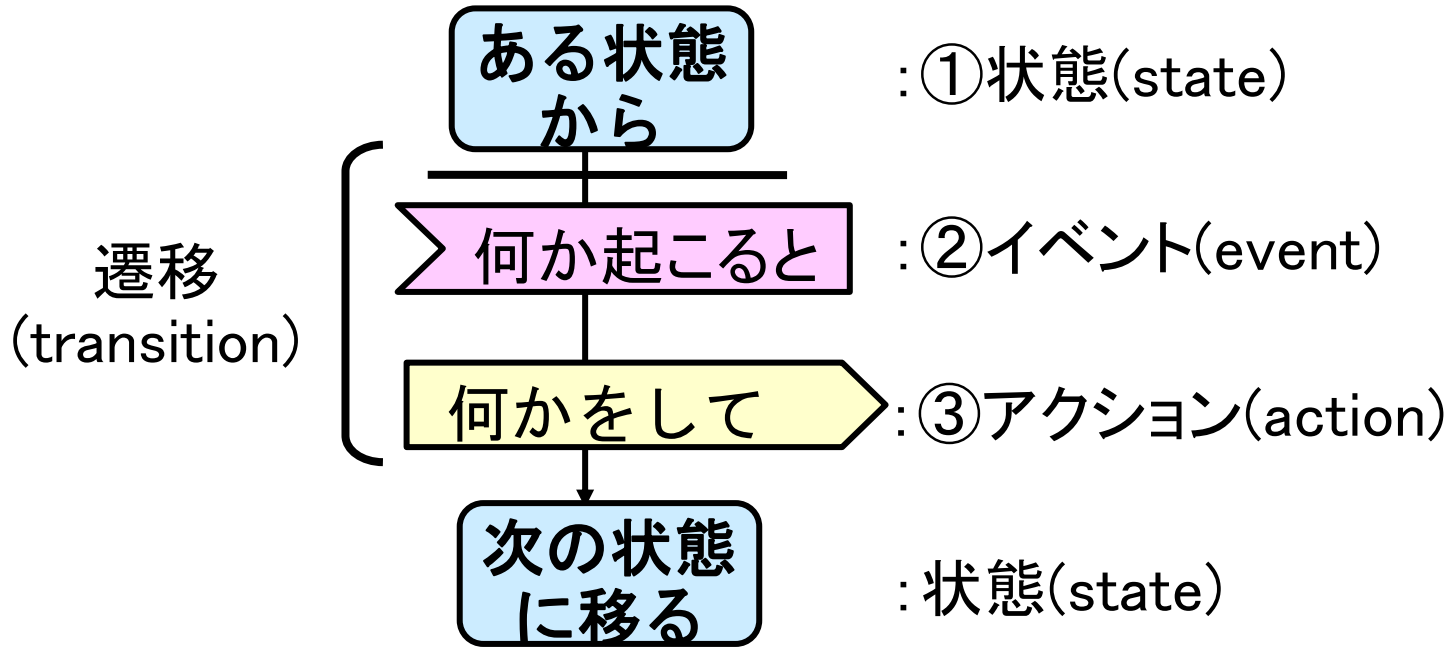
（1．1．2）状態遷移図

■“状態”の考え方で、スライド(1.1)の動き記すと、下図のようになります（この記法を、SDL(State Definition Language)と言います）。スライド(1.1)と同じ内容になっていることを確かめましょう。



（1．1．3）状態遷移図の要素

■状態遷移図では、（今のところ）3種類の要素があります。



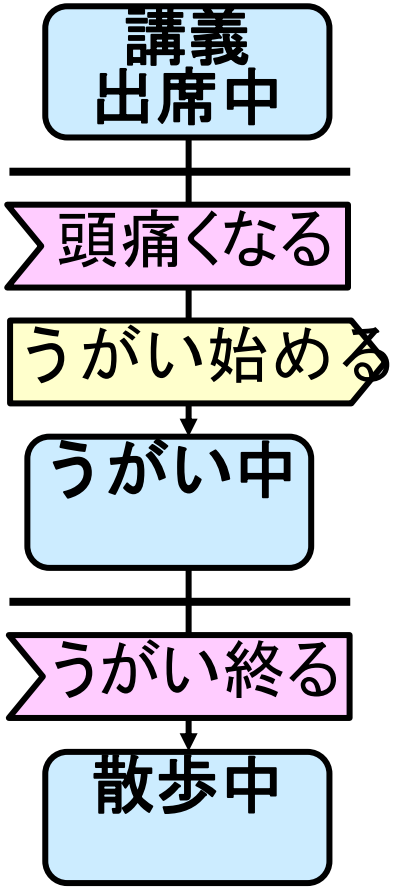
■意味から考えて、[状態]⇒[イベント]⇒[アクション]⇒[状態]の順に連なっており、他の順序はありません。但し、複数のアクションが続くのはOKです。

（1．1．4）うがい中？　－不要な状態－

■スライド(1.1.2)の状態遷移図では、“うがい中”とか“食事中”とかいった状態はありません。なぜでしょうか？

■例えば、“うがい中”という状態を右図のように作ったとしても、“うがい中”の前後には、「うがい始める」のアクションと「うがい終る」というイベントしかありません。これらは、“うがい中”という状態のために設けられたものであり、新たにアクション（イベント）が増えただけで、Aさんの生活を記述する上で役に立っていません。

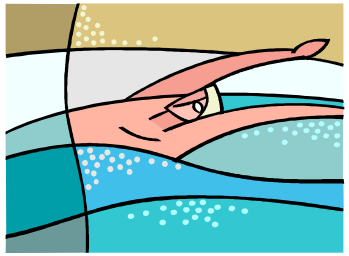
■このように不要な状態を定義しないようにしたほうが良いですね。

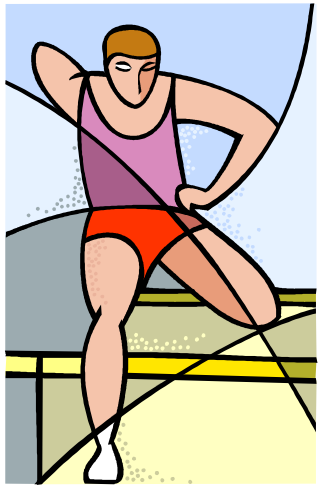


（1．2）もう一度、思考実験

■次の状況を、表にまとめる方法を考えましょう。

- 何人かの学生がおり、それぞれ名前がある。
- それらの学生、陸上部、水泳部、野球部に属している人がいる。
- それらの学生は、大垣市内、大垣市外、岐阜県外のいずれかに住む。





（1．2．1）さまざまな表

■方法1

名前	伊藤
所属	陸上
住所	市内

名前	佐藤
所属	陸上
住所	市外

名前	田中
所属	水泳
住所	県外

名前	鈴木
所属	野球
住所	市外

名前	斉藤
所属	野球
住所	県外

名前	中村
所属	水泳
住所	市内

■方法2

名前	所属	住所
伊藤	陸上	市内
佐藤	陸上	市外
田中	水泳	県外
鈴木	野球	市外
斉藤	野球	県外
中村	水泳	市内

■方法3

〈所属〉	
伊藤	陸上
佐藤	陸上
田中	水泳
鈴木	野球
斉藤	野球
中村	水泳

〈住所〉	
伊藤	市内
佐藤	市外
田中	県外
鈴木	市外
斉藤	県外
中村	市内

■方法4

〈陸上〉
伊藤
佐藤

〈水泳〉
田中
中村

〈野球〉
鈴木
斉藤

〈市内〉
伊藤
中村

〈市外〉
佐藤
鈴木

〈県外〉
田中
斉藤

⇒　どれでも同じ情報が得られますね。どれが良いのでしょうか？

（1．2．2）区別

■どれが良いという話には、当面触れません。

- [方法1]は、オブジェクト指向風、[方法2，3]はデータベース風、[方法4]は集合論風と言えるかも知れません。
- 4つの方法以外にも、表の作り方はあります。

■ここで考えることは。。。

- スライド(1.1)シリーズで扱った状態は、プログラムの中では変数として保持されることは想像がつくと思います。
- ここのスライド(1.2)シリーズで扱った表も、やはりプログラムの中では変数として保持されることは想像がつくと思います。
- それでも、(1.1)と(1.2)とには、何か違いがあるように思われます。ここで問題にする違いは、「(1.1)はひとつの変数だが、(1.2)は複数の配列変数だ」というような違いではありません。(1.1)のデータでも、複数の人を考えれば、配列になります。

（1．3）呼び分け

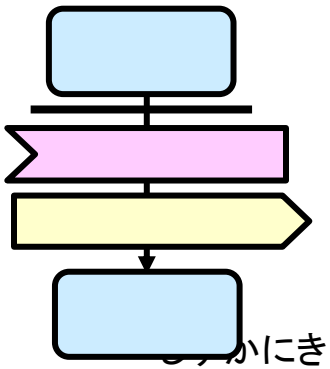
■前記(1.1),(1.2)を呼び分ける名前には、衆目一致して認めるものが無いようです。

■次のような呼び分けは、割と賛同が得られるかも知れません。

- (1.1)のようなものは“状態”、(1.2)のようなものは“データ”
- しかしながら、2つを対比させないならば、(1.1)のようなものに対しても、“データ”と呼ぶことは普通です。

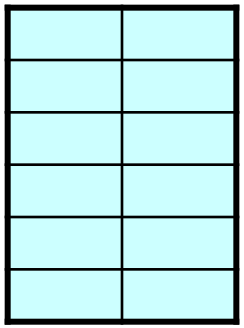
■今講座では、少し苦しいのですが、必要な場合は次のような呼びわけを行います（一般的ではないので注意してください）。

(1.1)状態、もしくは、
状態データ



データと言えば、
両方を指す

(1.2)情報データ、
もしくは、情報



（2）Jude

■ここからしばらく((2),(3),(4))は、“状態”のみを扱います。

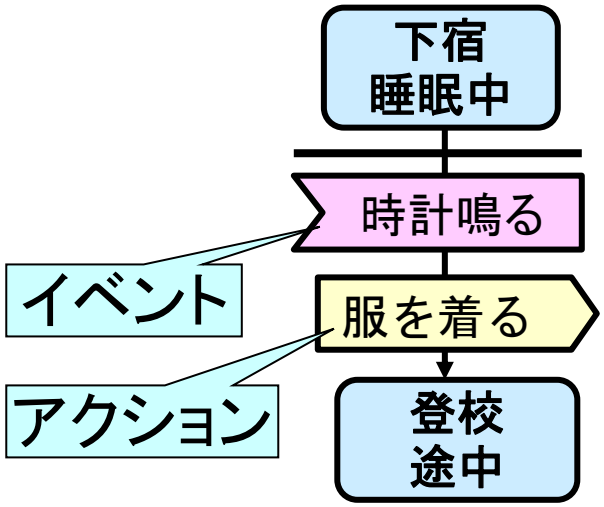
■まず、状態遷移図を描くツールの使い方を覚えましょう。ここでは、フリーソフトの“Jude”を用います（http://objectclub.esm.co.jp/Jude/jude.html）。

- Judeは、日本でのプロジェクトにより開発されています。
- Judeは、本来、UML（オブジェクト指向モデリングを行うときに標準的に用いられるモデリング言語）のためのツールです。ここでは、そのうちの状態遷移図(ステートチャート)等を描く機能を利用します（UMLについては、別講座で学びます）。
- Judeのようなツールを、CASE(Computer Aided Software Engineering)ツールと呼びます。

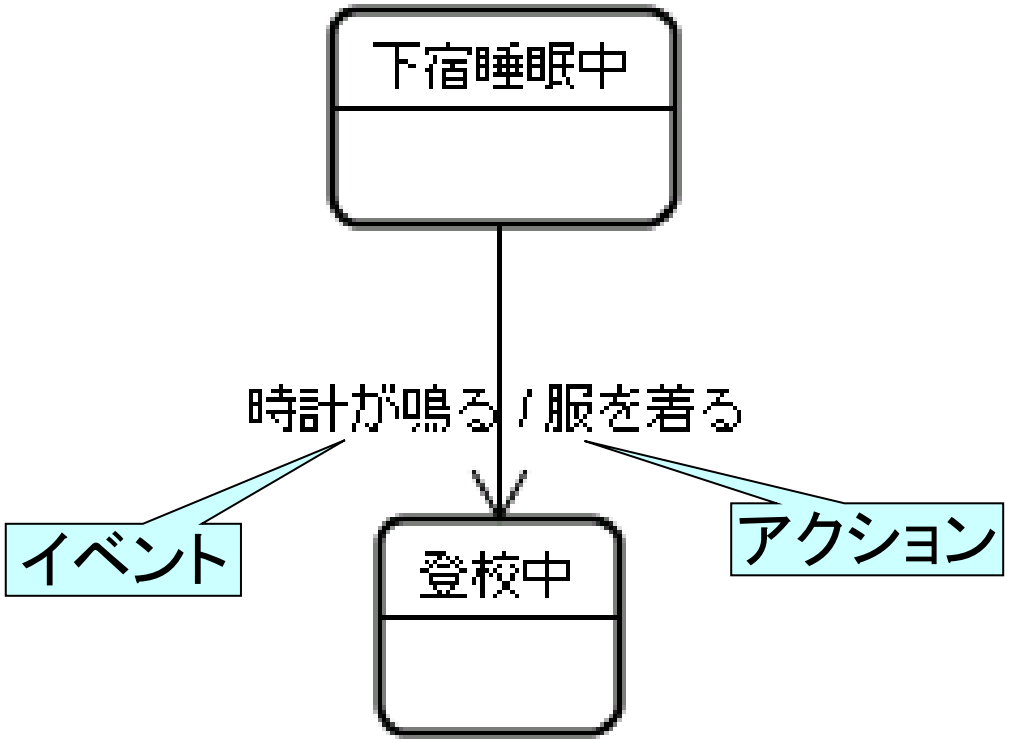
（2．1）Judeでの状態遷移図の記法

■JudeはUMLのツールですから、状態遷移図（ステート・チャート）の記法は、SDLとは異なります。

＜SDL＞



＜UMLのステートチャート＞



■Judeでは、イベントとアクションとの間の区切りの“スラッシュ（”/”）”で区切られています。

（2．2）Judeの起動

■学内のWindowsPCは、Judeがインストール済みです。

- 次のショートカットで起動します。

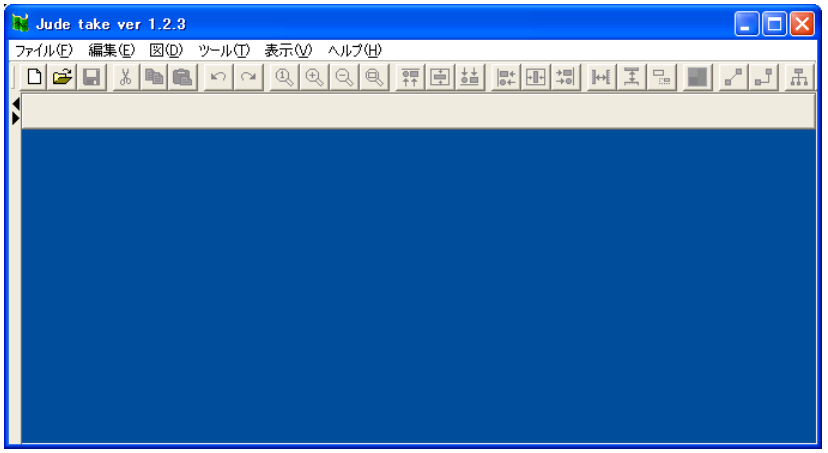
  [スタート]

  –[すべてのプログラム]

  –[アプリケーション]

  –[Jude]–[Jude]

- 右のような画面が現れます。



■みなさんのパソコンでも、Judeをインストールできます。

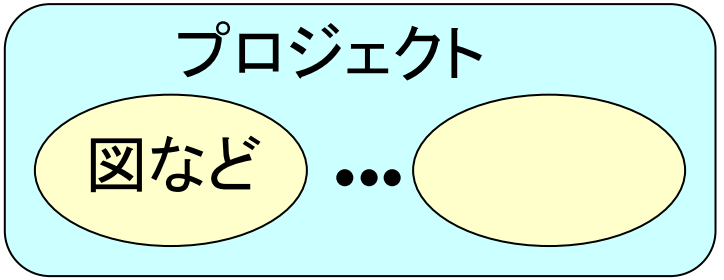
- Judeのインストールについても、次のサイトを参考にしてください。

http://objectclub.esm.co.jp/Jude/jude.html



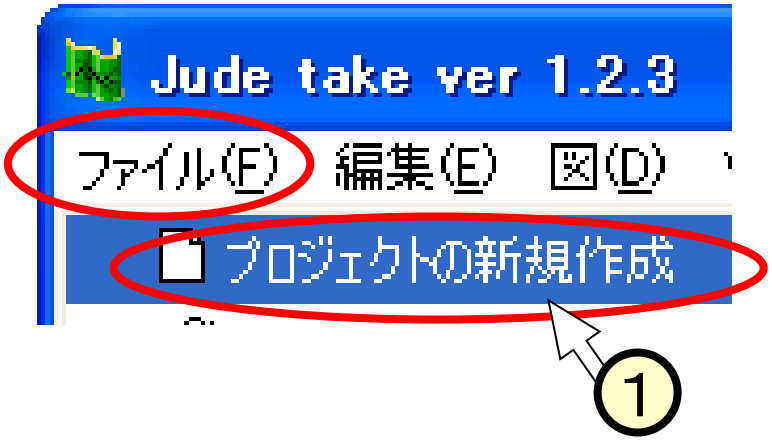

（2．3）プロジェクトの新規作成

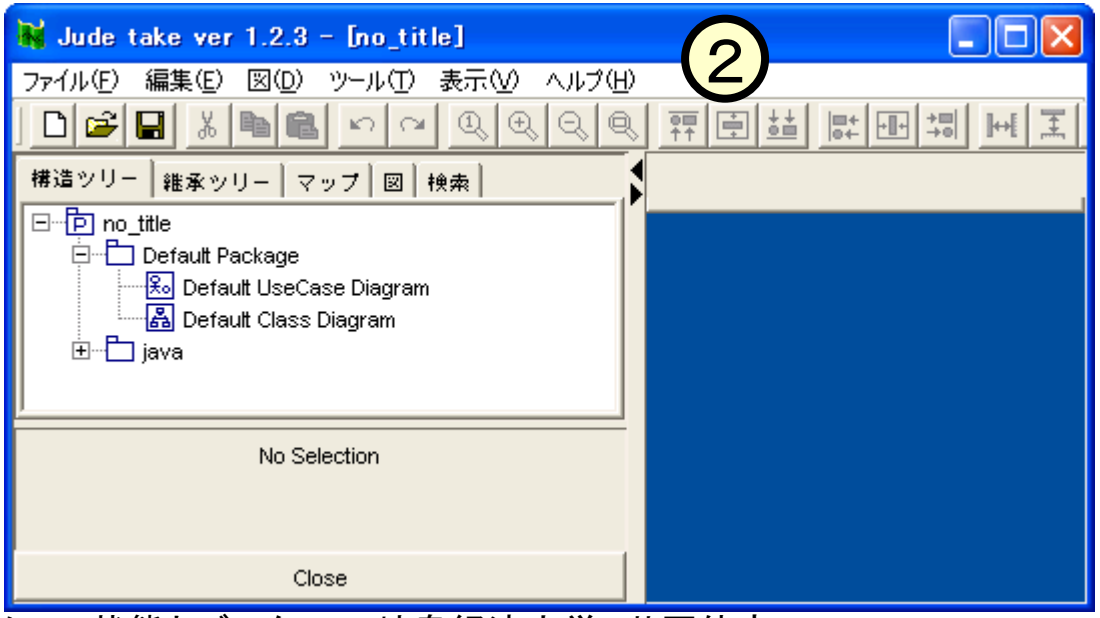
■Judeで作成される図などは、
プロジェクトという単位で
管理されます。



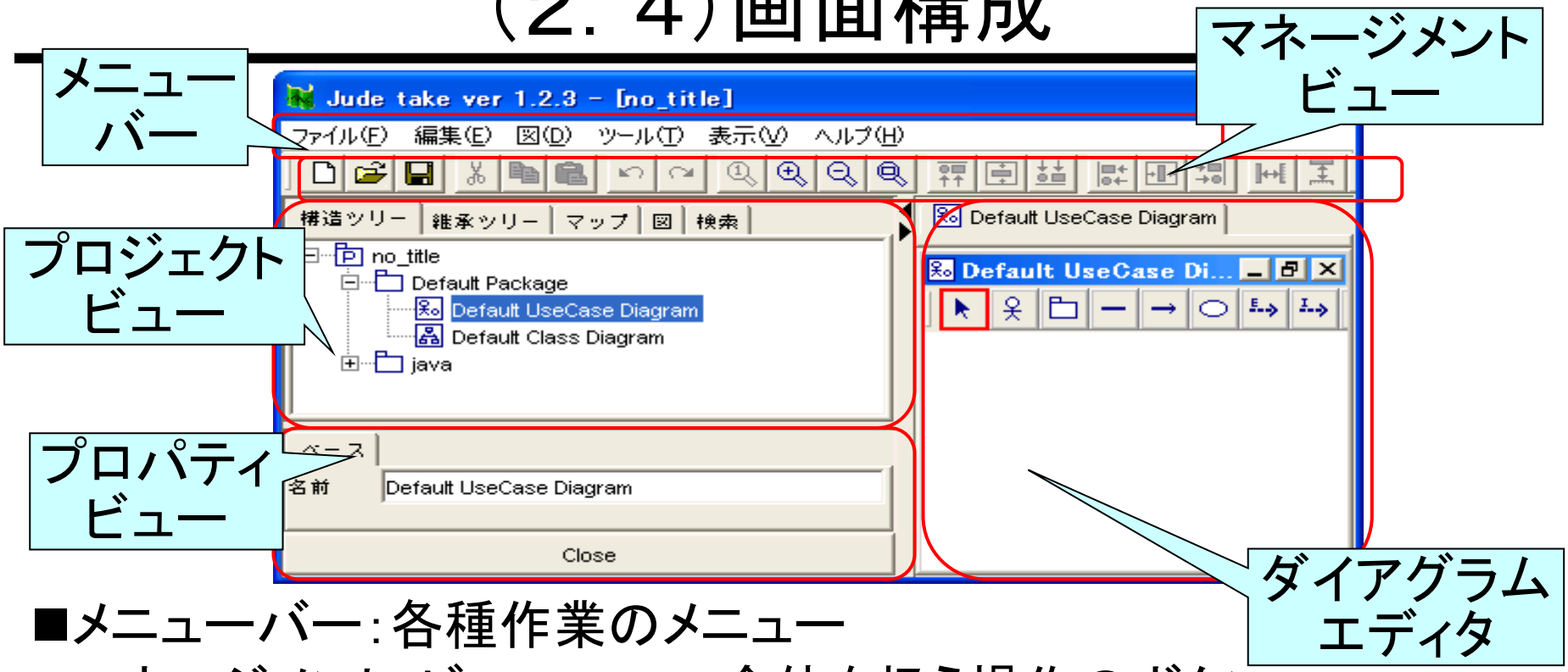
■最初にこのプロジェクトを作成します。

- メニューから、[ファイル]–[プロジェクトの新規作成]をクリック（①）します。②のような画面となります。





（2．4）画面構成



■メニューバー：各種作業のメニュー

■マネージメント・ビュー：Jude全体を扱う操作のボタン。

■プロジェクト・ビュー：上部のタブを切り替えることで、さまざまな方法でプロジェクトの構成を示します。ここで選んだ図をダイアグラム・エディタ上で操作することが出来ます。

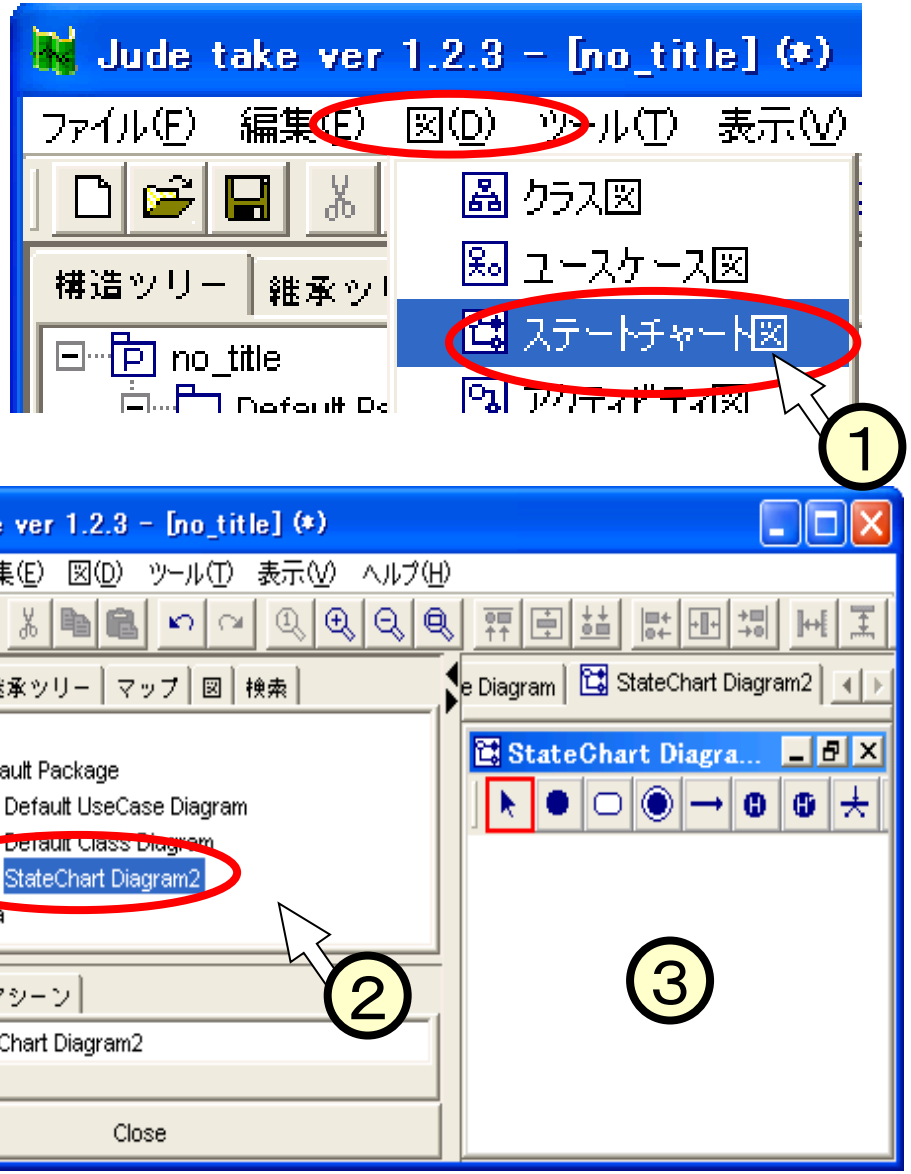
■ダイアグラム・エディタ：ここでダイアグラム（図）を作成する。

■プロパティ・ビュー：ダイアグラム（図）中の要素への操作を行います。

（2．5）新規ステートチャートを開く

■メニューより、[図]–[ステートチャート]をクリック（①）します。

■ツリー画面に新しいダイアログ（図）が表示され、これをクリック（②）してフォーカスする（薄紫色になる）と、編集画面（③）にて図の作成が可能になります。



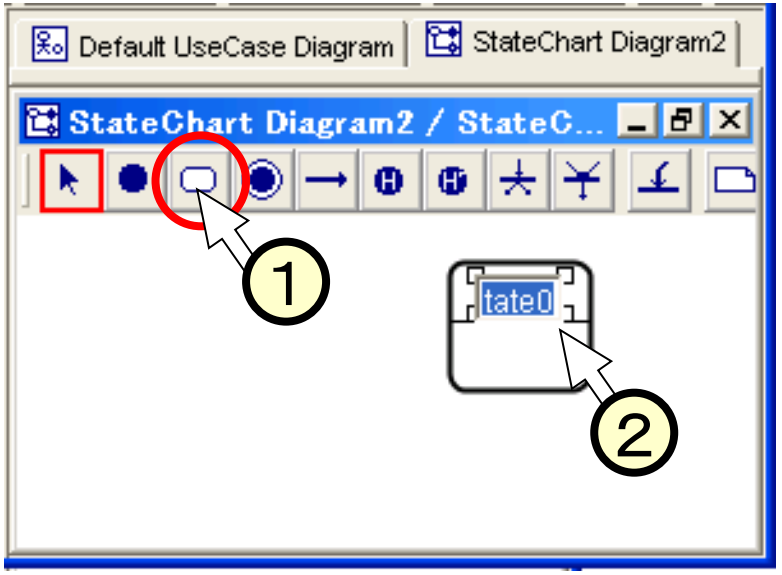
（2．6）状態要素の配置

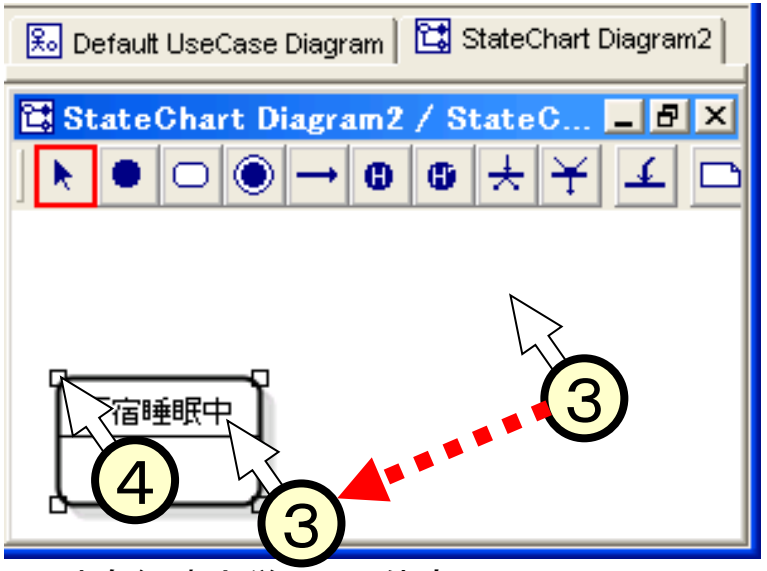
■ツールバーの[状態]をクリック（①）して、編集画面上をクリック（②）します。

■状態要素が、名前の入力待ちの形で配置されます。ここで名前の入力が可能ですが、後から名前の欄をダブルクリックして入力することも出来ます。

■ドラッグ（③）することで、移動させます。また、要素の四隅のつまみをドラッグ（④）して、大きさを調整します。

■フォーカスされた状態（＝四隅につまみが表示）で[Delete]キー押下すると、要素は削除されます。

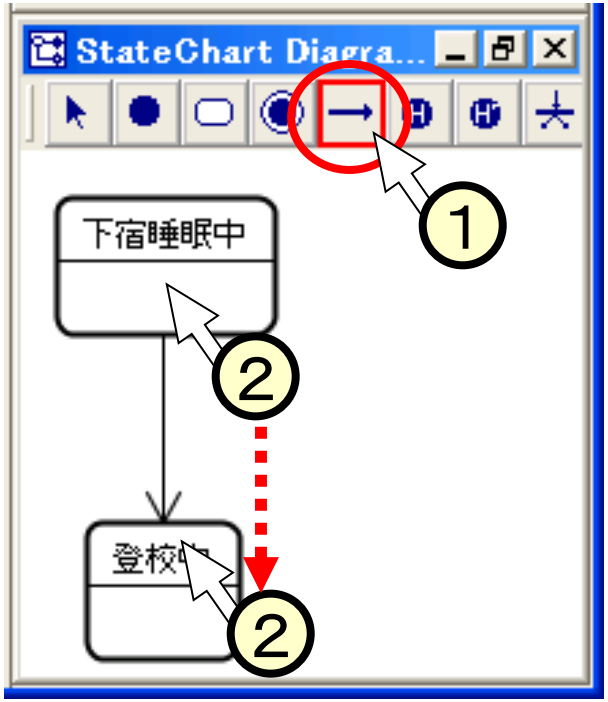




（2．7）遷移要素の配置

■スライド(2.6)のようにして、2つの状態要素を作ったとします。この2つの間に、遷移要素を付け加えます

■ツールバーの遷移要素をクリック（①）して、遷移元の状態要素から遷移先の状態要素へ、ドラッグ（②）します。



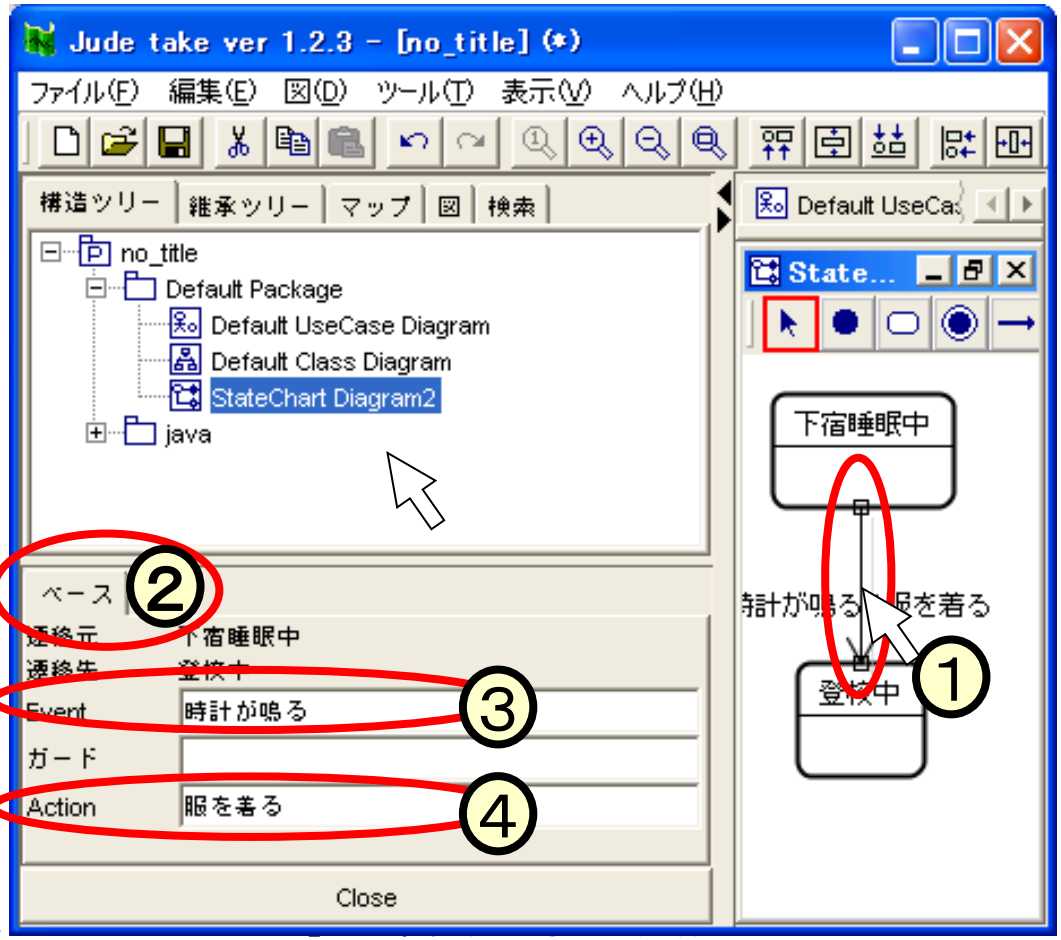
（2．8）イベント・アクションの入力

■配置された遷移要素をクリック（①）してフォーカスすると、プロパティ・ビューにその特性が表示されます。

■ベースタグ（②、これしかありません）中にて、次の入力を行います。

- Event :
  イベントの入力（③）
- Action:

アクションの入力（④）

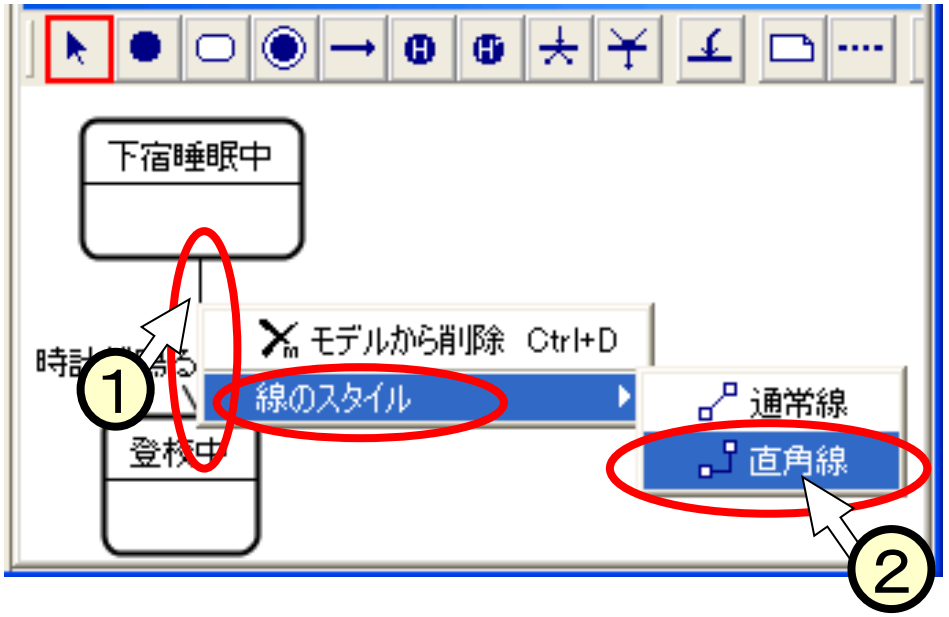


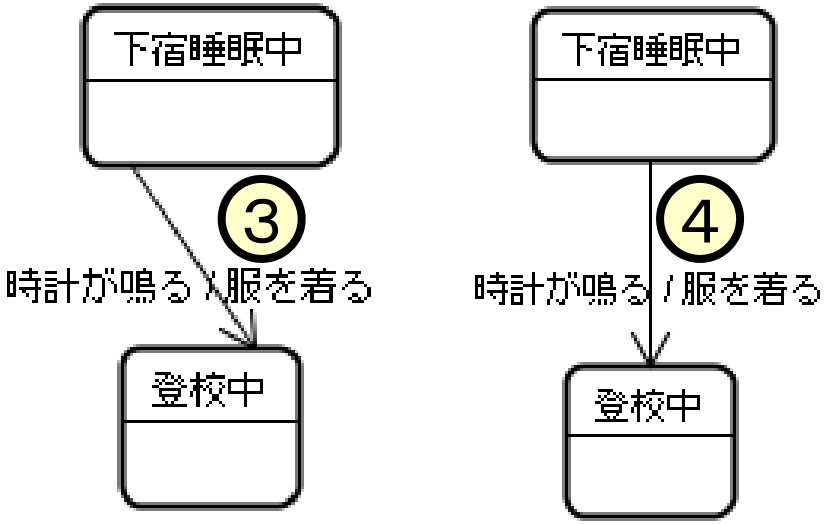
「しずかにきしれ四輪プログラム

（2．9）通常線・直角線

■遷移要素を右クリック（①）すると、ポップアップメニューの[線のスタイル]にて、[通常線]、[直角線]（②）のいずれかが選択できます。

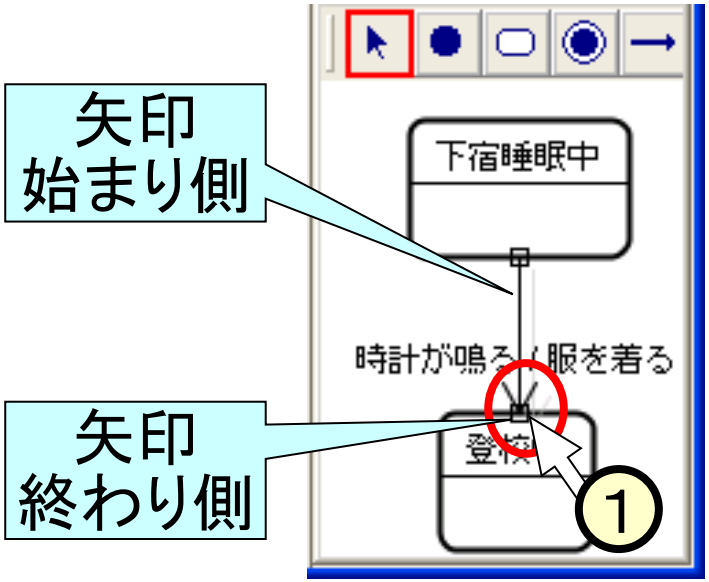
■[通常線]では、右図③のような斜めの配置が可能ですが、これを[直角線]を指定すると、右図④のように必ず直角に配置されます。





（2．10）遷移要素の配置の変更

■直角線の遷移要素の配置を変更する際には、フォーカスした際に現れたつまみのうち、矢印の終わり側のつまみをドラッグ（①）して移動させてください（矢印の始まり側では動きません）。

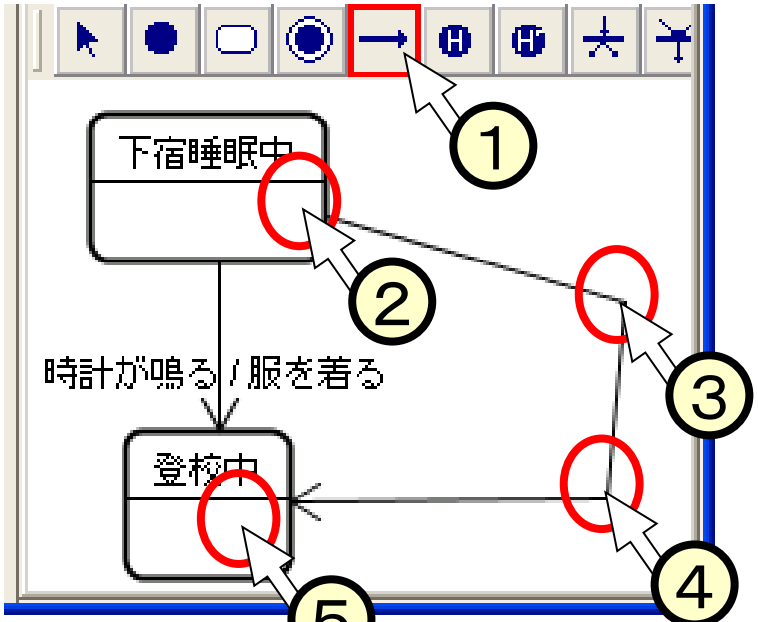


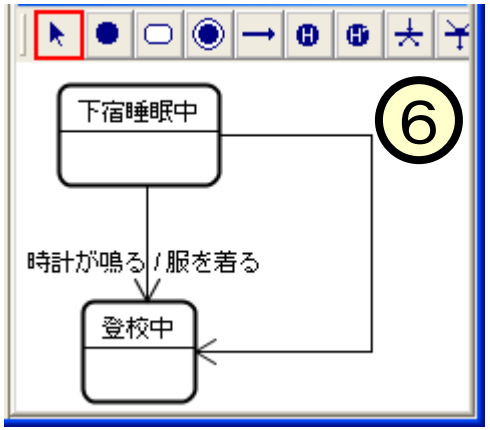
（2．11）遷移要素の折れ線での描画

■遷移要素は折れ線でも描画できます。

- ツールバーの遷移要素をクリック(①)します。
- 遷移元状態をクリック(②)したのち、折れ線の各頂点をクリック(③、④)していき、最後に遷移先状態をクリック(⑤)します。

■この折れ線に対して、[直角線]を指定すると、⑥のようになります。

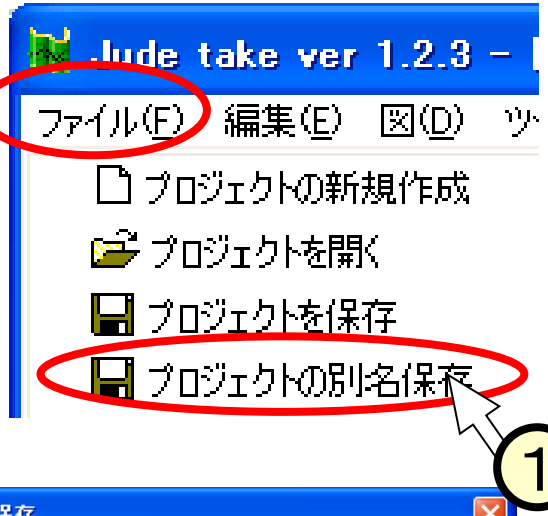




（2．12）ファイルのセーブ

■ファイルのセーブは、[ファイル]–[プロジェクトの別名保存]を選択し（①）、「保存」のダイアログ（②）を用いて普通に保存出来ます。

■この時指定したファイル名は、プロジェクト名前になり、プロジェクトビュー上に表示（③）されます。



（2．13）図に名前を変更する

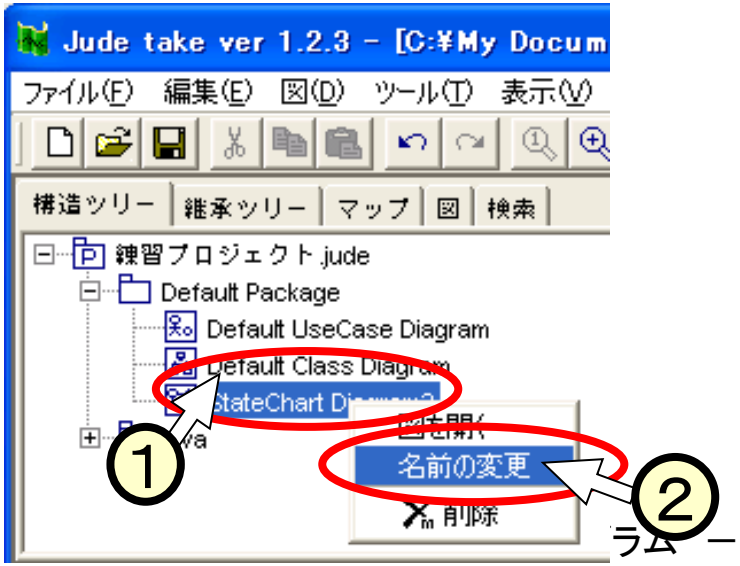
■これから複数の図を書いていく際、図に名前を付けておかないと混乱が生じます。

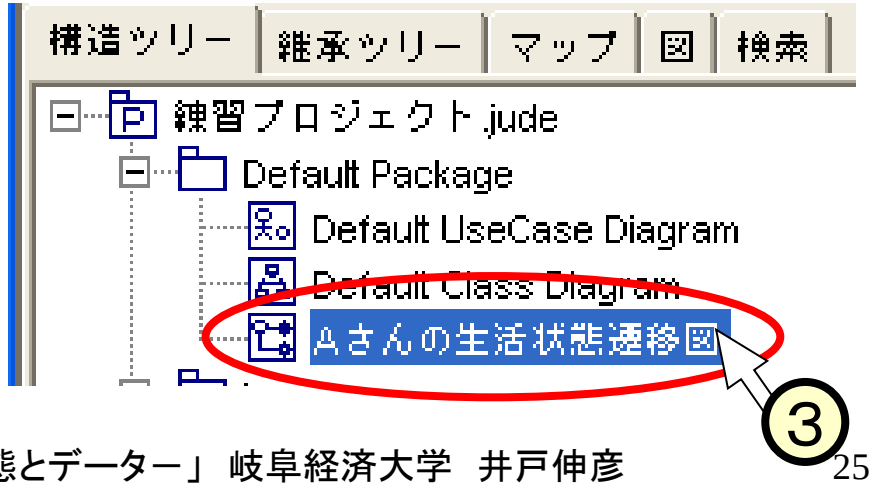
■プロジェクト・ビュー上で、図の名前を変更します。その方法は、普通のWindowsフォルダ上と同じです。

- 名前を変更する図を右クリック（①）して、ポップアップメニューから[名前の変更]をクリック（②）する。

（もしくは図名のところを間隔を置いて2度クリックする）

- 入力が可能な状態になるので、名前を入力（③）する。



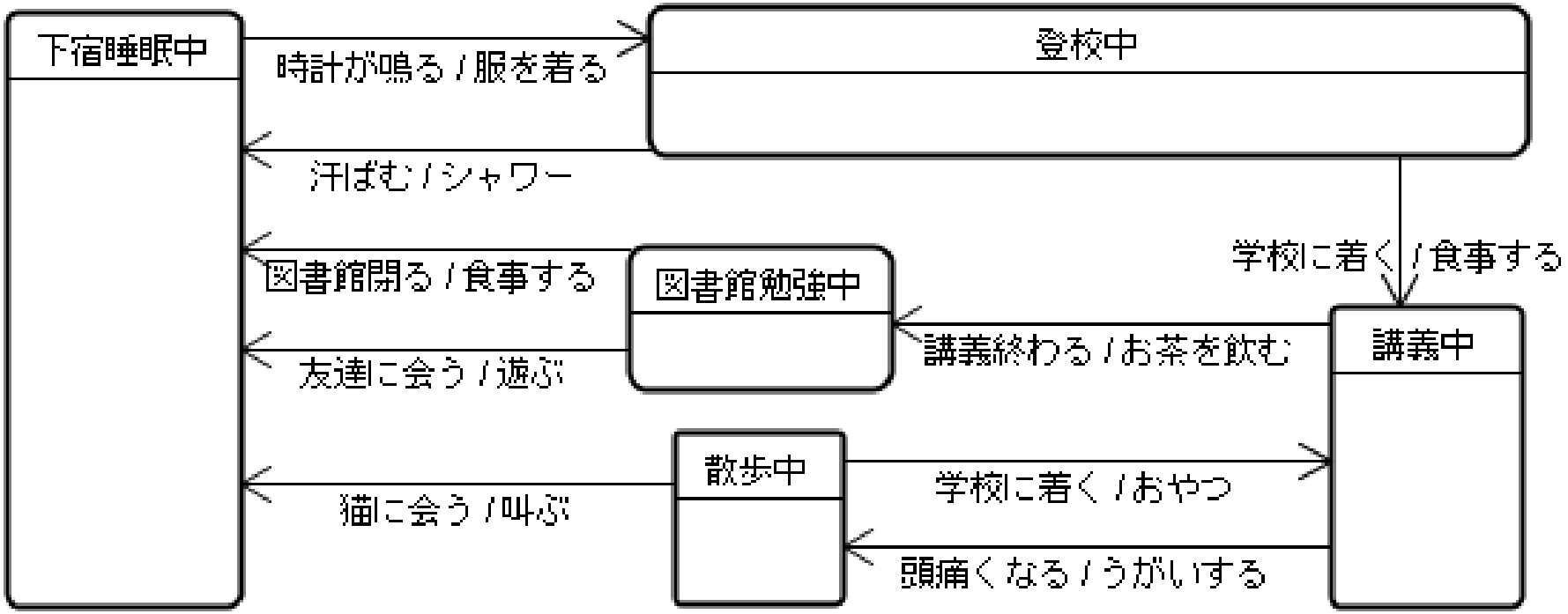


（2．14）演習（課題2－1）

■課題2－1：スライド(1.1.2)の状態遷移図を、Judeにて作成してください。

- 配置した状態要素を移動させても、遷移要素は関係を保って自動的に移動します。見やすいように配置を工夫してください。

■解答



- 遷移が横になるように状態を配置することがコツのようです。

（2．15）演習（課題2－2）

■次に記した消防士の状態遷移図を作成してください。

- 休み中に、出勤時刻になると、制服に着替えて、勤務中となる。
- 休み中に、緊急呼び出しを受けると、制服に着替え、消火活動中になる。
- 勤務中に、サイレンがなると、消防車に乗り込み、消火活動中となる。
- 勤務中に、昼飯時になると、食堂に出かけ、食事中となる。
- 勤務中に、退勤時刻になると、制服を脱いで、休み中となる。
- 食事中に、サイレンがなると、食事を止め、消防車に乗り込み、消火活動中となる。
- 消火活動中、鎮火すると、本部に報告して、勤務中となる。

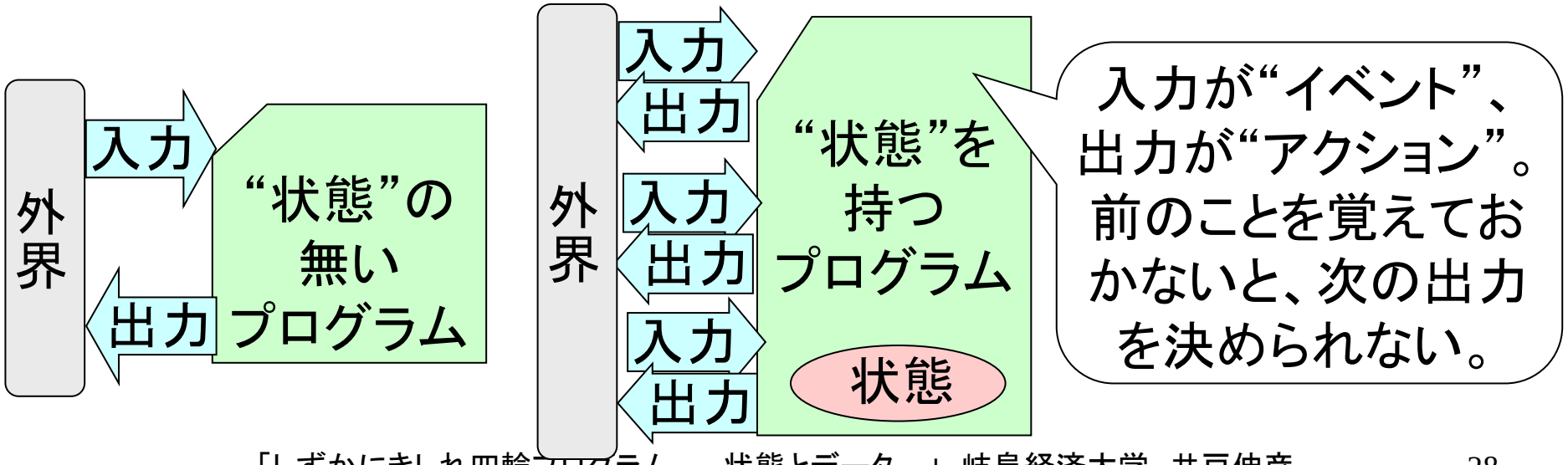
（3．1）“状態”はいつ使うのか？

■「私は何度もプログラムを書いたけど、“状態”なんて使ったことが無い」と思われるかも知れません。

- 一度入力が与えられて、それを処理して出力するだけの小さなプログラムでは、“状態”の出番は無いかも知れません。

■小さなプログラムでも、外界とインタラクティブに動く（＝やりとりをする）プログラムでは、“状態”が生じます。

（自身以外のプログラムも、外界と考えます。）



（３．２）例１：東西南北

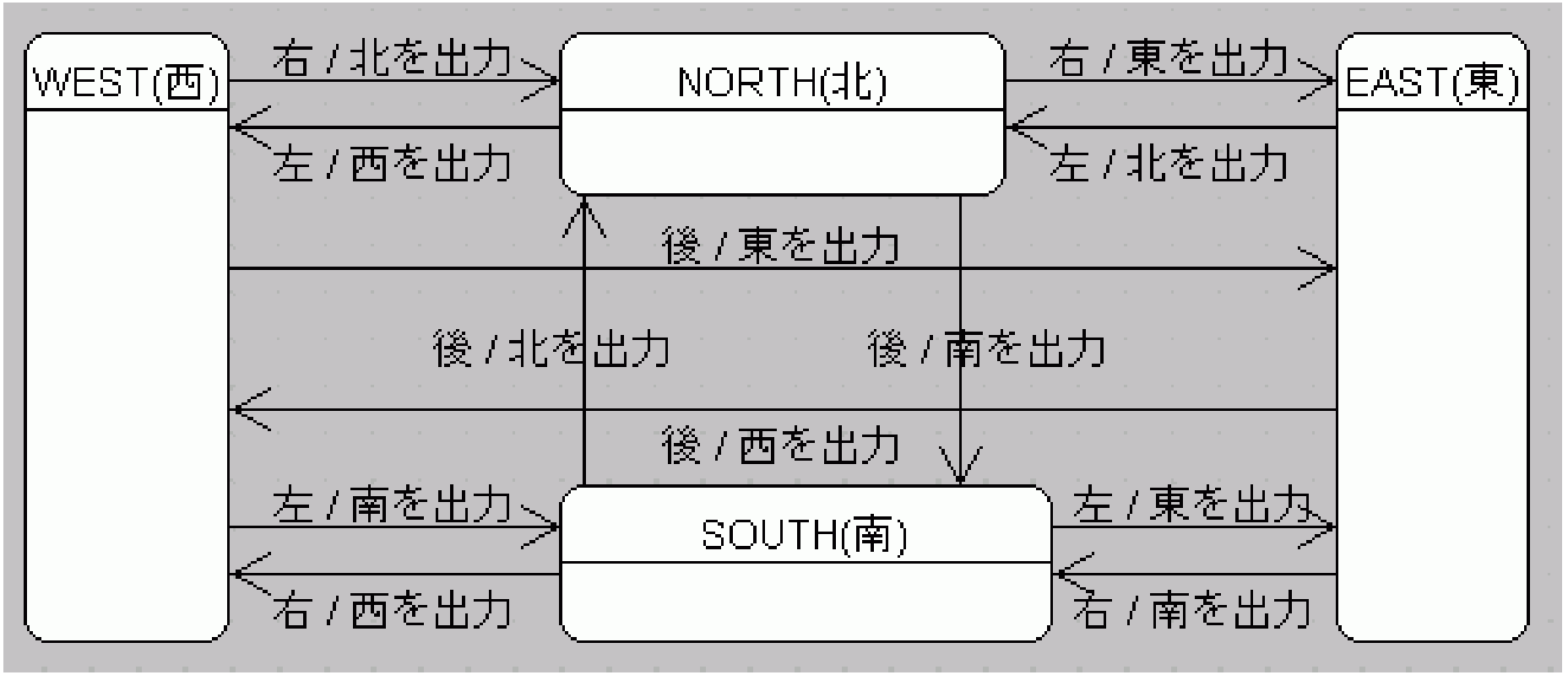
■状態を持つ例として、次のようなプログラムを考えます。
- プログラムが起動されると、最初、（人は）東を向いています。
- “r”(right)と入力されると、右に方向を変えます。
- “l”(left)と入力されると、左に方向を変えます。
- “b”(back)と入力されると、後ろに方向を変えます。
- 方向を変えた後、今どちらに向いているかを出力します。
- “q”(quit)と入力されると、プログラムは終了です。

■プログラムは、後のスライドに示します。
- Javaで書かれています。
- ファイル名は、“News.java”とします。

■**eclipse**上で実行してみます。

（３．２．１）状態遷移図

■状態遷移図（ステートチャート）は、下図のようになります。



（3．2．2．1）プログラム　－1/3－

```
import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;
public class News{
   static final int EAST =0;
   static final int NORTH=1;
   static final int WEST =2;
   static final int SOUTH=3;
public static void main(String args[]) throws IOException{
     int state=EAST;
     String ss;
     BufferedReader kbd =
      new BufferedReader(new InputStreamReader(System.in));
     while((ss = kbd.readLine())!=null){
       char event=ss.charAt(0);
       if(event=='q'){ break;}
```

“EAST”と書けば、“0”を意味するようにします。他も同様。

これが“状態”です。最初は東です。

入力を読み取りながら繰り返し。

“q”が入力されたら、繰り返しを止めます。

文字列を読み取ってイベント（トリガー）とします。

（3．2．2．2）プログラム　－2/3－

状態が東の場合

イベントが“r”の場合

アクション（メッセージ出力）

次の状態

北、西、南の場合も
同様に状態遷移します。

```
if(state==EAST){
  if(event=='r'){
    System.out.println("I turned to the SOUTH");
     state=SOUTH;
  }else if(event=='l'){
    System.out.println("I turned to the NORTH");
    state=NORTH;
  }else if(event=='b'){
    System.out.println("I turned to the WEST");
    state=WEST;}
}else if(state==NORTH){
  if(event=='r'){
    System.out.println("I turned to the EAST");
    state=EAST;
  }else if(event=='l'){
    System.out.println("I turned to the WEST");
    state=WEST;
  }else if(event=='b'){
    System.out.println("I turned to the SOUTH");
    state=SOUTH;
  }
```

(3. 2. 2. 3)プログラム　－3/3－

```
        }else if(state==WEST){
          if(event=='r'){
              System.out.println("I turned to the NORTH");
              state=NORTH;
          }else if(event=='l'){
              System.out.println("I turned to the SOUTH");
              state=SOUTH;
          }else if(event=='b'){
              System.out.println("I turned to the EAST");
              state=EAST;
          }
        }else if(state==SOUTH){
          if(event=='r'){
              System.out.println("I turned to the WEST");
              state=WEST;
          }else if(event=='l'){
              System.out.println("I turned to the EAST");
              state=EAST;
          }else if(event=='b'){
              System.out.println("I turned to the NORTH");
              state=NORTH;}
        }
      }
    }
}
```

（3．2．3．1）プログラム　ー前半ー

```
<html>
<head><title>s_3_2_3</title></head>
<body onload='setInterval(main,100);'>
<script type="text/javascript">
const EAST= 0;
const NORTH=1;
const WEST= 2;
const SOUTH=3;
var state=EAST;
function main(){
  var in_c=document.getElementById('text0');
  var out_c=document.getElementById('p0');
  var event=in_c.value;
  if(event=='')return;
  if(state==EAST){
    if(event=='r'){ out_c.innerText='南'; state=SOUTH;
    }else if(event=='l'){ out_c.innerText='北'; state=NORTH;
    }else if(event=='b'){ out_c.innerText='西'; state=WEST;
    }
  }else if(state==NORTH){
    if(event=='r'){ out_c.innerText='東'; state=EAST;
    }else if(event=='l'){ out_c.innerText='西'; state=WEST;
    }else if(event=='b'){ out_c.innerText='南'; state=SOUTH;
    }
```

100ms毎にmainを起動します。

“EAST”と書けば、“0”を意味するようにします。他も同様。

これが“状態”です。最初は東です。

入力欄を取得。

出力欄を取得。

入力文字を取得。

入力無しなら終了

(3．2．3．2)プログラム　ー後半ー

状態が西の場合

入力が“r”(=右)ならば、
“北”を出力して、
状態は”NORTH”になる

```
  }else if(state==WEST){
    if(event=='r'){ out_c.innerText='北'; state=NORTH;
    }else if(event=='l'){ out_c.innerText='南'; state=SOUTH;
    }else if(event=='b'){ out_c.innerText='東'; state=EAST;
    }
  }else if(state==SOUTH){
    if(event=='r'){ out_c.innerText='西'; state=WEST;
    }else if(event=='l'){ out_c.innerText='東'; state=EAST;
    }else if(event=='b'){ out_c.innerText='北'; state=NORTH;
    }
  }
  in_c.value='';
}
</script>
<form>
<input type='text' id='text0' />
</form>
<p id='p0'></p>
</body>
</html>
```

入力欄を空にする

入力欄

出力欄

（３．２．４）動かして見る

■東西南北のプログラムを、井戸のネットワークドライブからダウンロードして、動かしてみます。

- 井戸のネットワークドライブから、次のファイルをコピーしてください。
  - ◆News.java
- eclipseへインポートして実行します。次のスライドを参照してください。
  - ◆月に吠える　－eclipseによるJavaアプリケーション作成－　スライド（６）ファイルのインポート

（3．2．5）演習番外編

■この例では、状態遷移に規則性があるので、スライド(3.2.2)(3.2.3)のような状態遷移を用いたプログラム以外の解法があります。

■スライド(3.2.2)(3.2.3)のif文の部分(すなわち、**if(state=…){ if(event=…}**の部分)を、次のように書き換えても、全く同じ動きをします。

```
int change=0;
if(event=='r'){ change=3; }
else if(event=='l'){ change=1; }
else if(event=='b'){ change=2; }
state=(state+change) % 4;
```

■演習番外編：上記のプログラムがどういう考えで動くかを説明してください。

（３．３．１）例2：フォーマット

■次のようなプログラムを考えます。

- 左上側のようなファイルを読み込んで、標準出力に左下側のような出力を行う。
- ファイル中の一人分の記録は、“n”、”b”、”h” の順に現れる。順番が狂っていることを発見した場合、プログラムはエラーを出力し、後の動作は保証しない。
- ファイルには、一人ごとに、n（name）の行は必ずありますが、その他の行、”b”、”h”は無い場合もある。
- 区切りの記号は、右表の通り。

```
n:井戸　伸彦
b:1960年5月14日
h:岐阜県
n:元　ちとせ
h:沖縄県
```

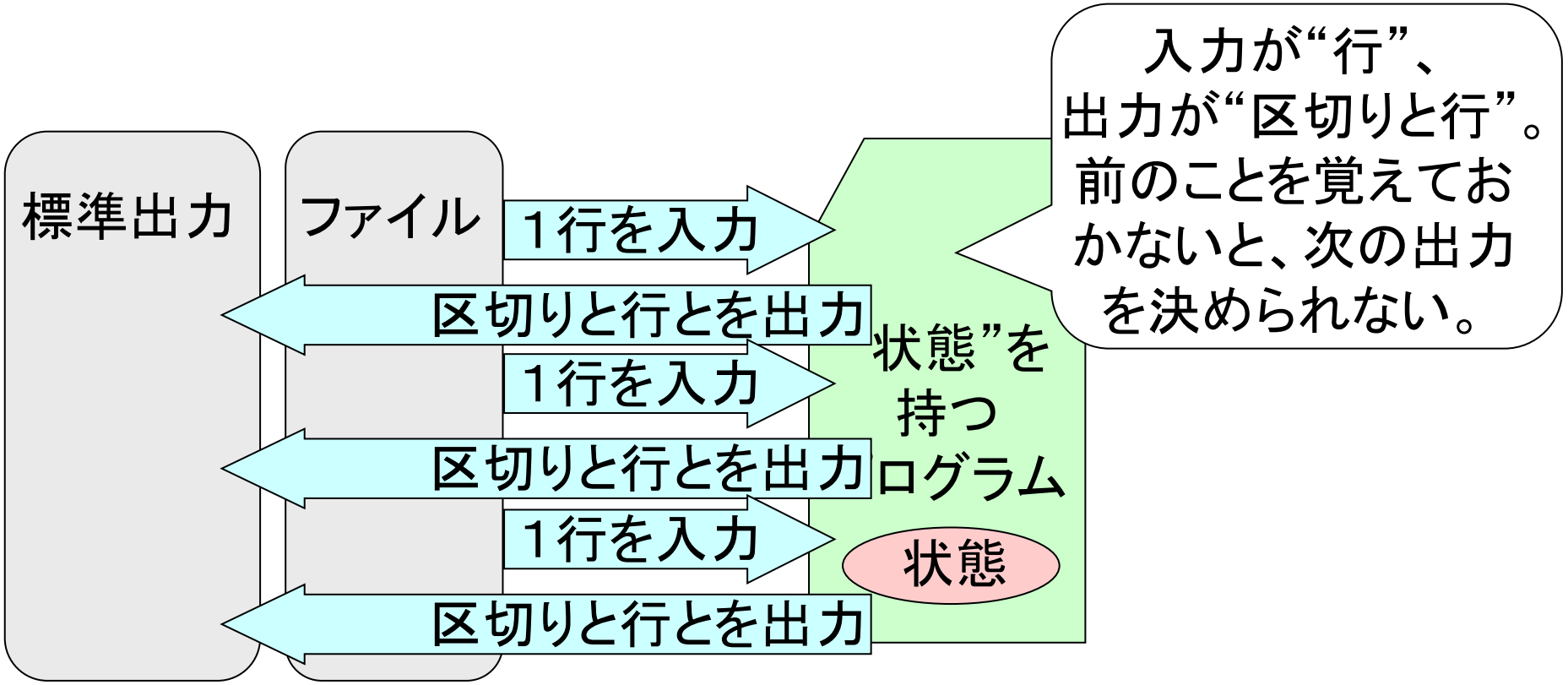
前	後	区切り
	n	====
n	b	----
n	h	----

```
====
n:井戸　伸彦
----
b:1960年5月14日
h:岐阜県
====
n:元　ちとせ
----
h:沖縄県
```

（３．３．２）なぜ状態が必要か？

■このプログラムは外界とやりとりする訳ではありませんが、次のような状態遷移と考えることが出来ます。

- 直前に何を入力したかにより、状態を持つ。
- １行づつを入力し、出力と次状態とを決める。



（3．3．3）状態遷移図

■次の3つの状態で考えます。

- state_n

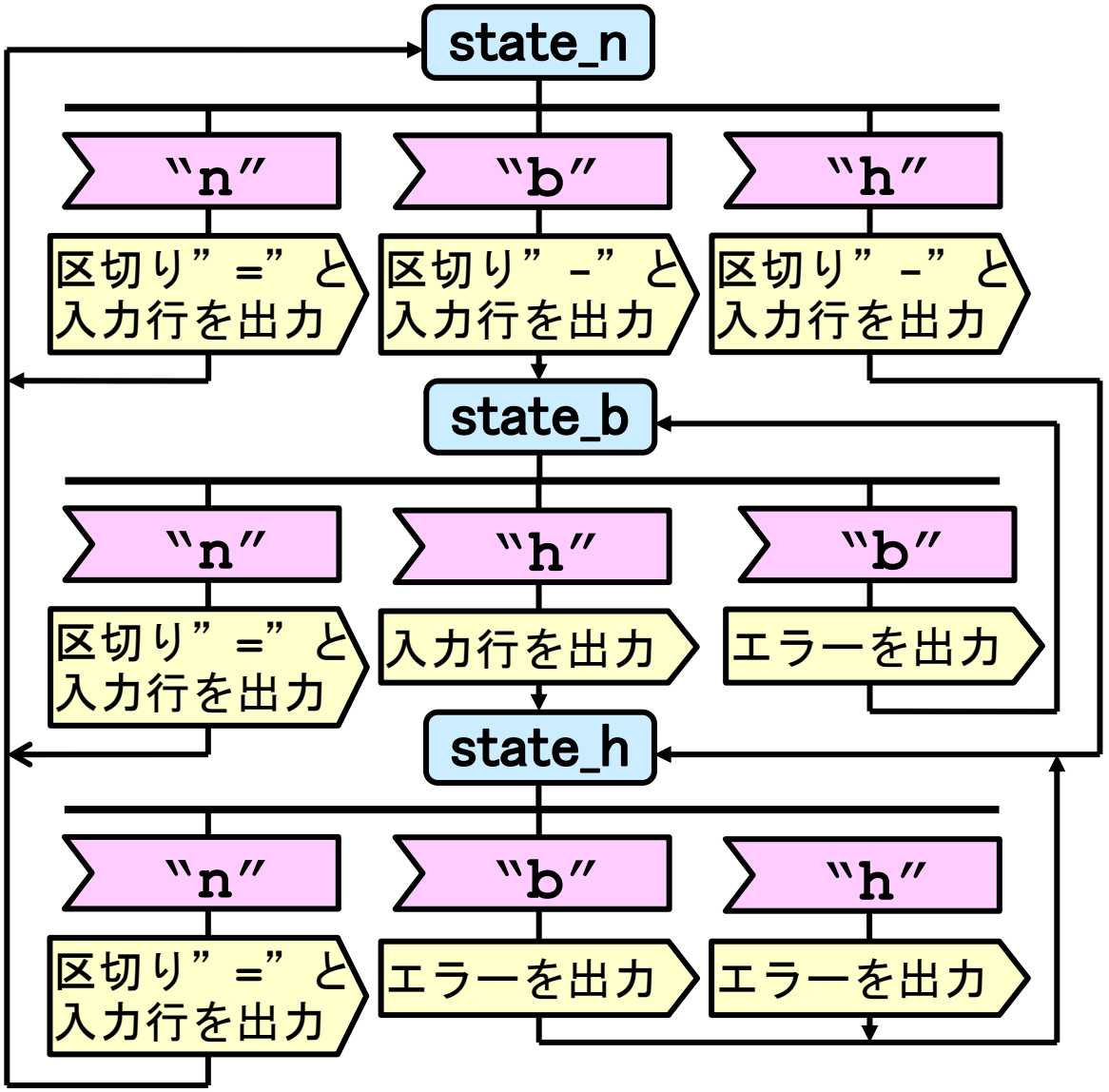
  直前に“n”を入力した状態

- state_b

  直前に“b”を入力した状態

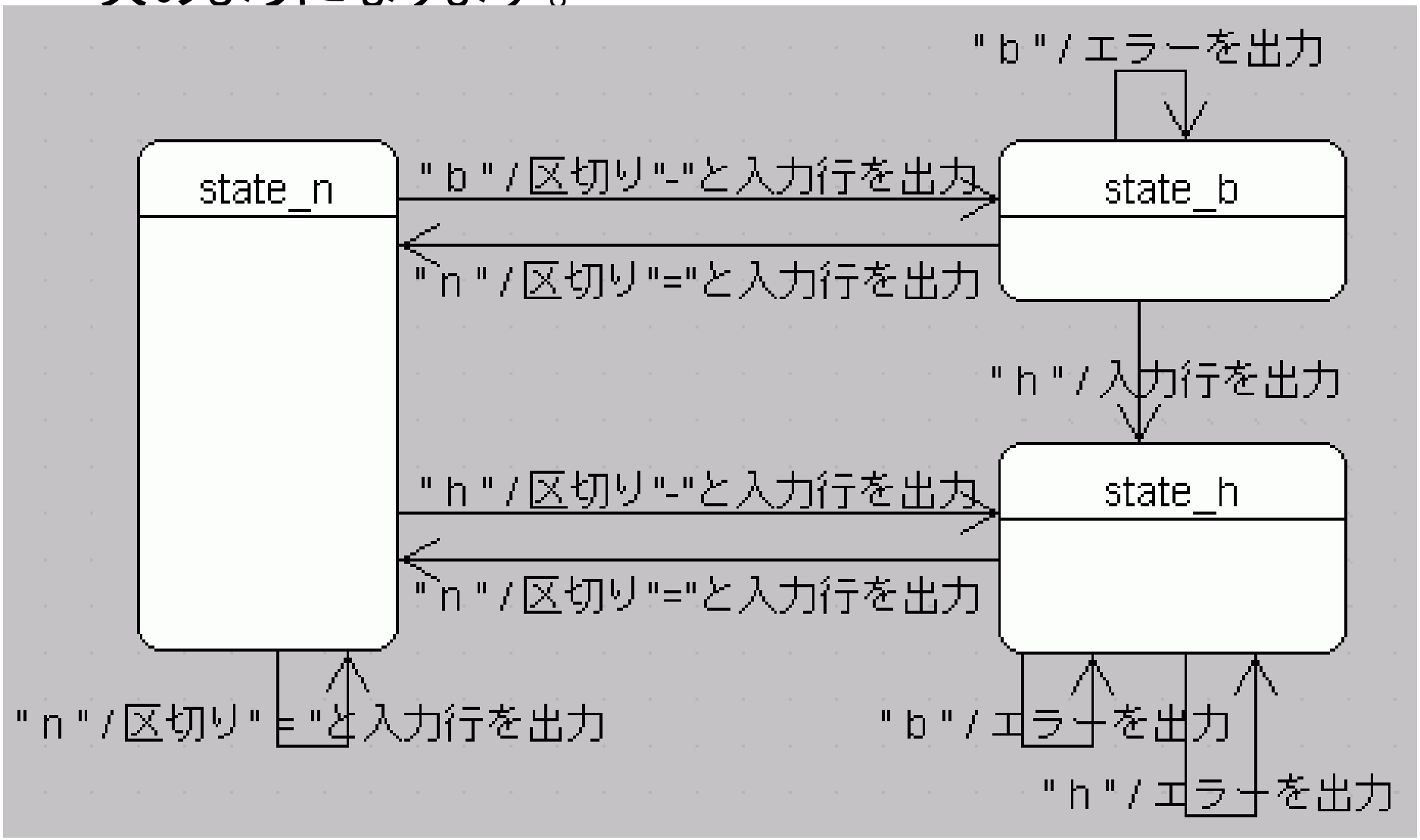
- state_h

  直前に“h”を入力した状態

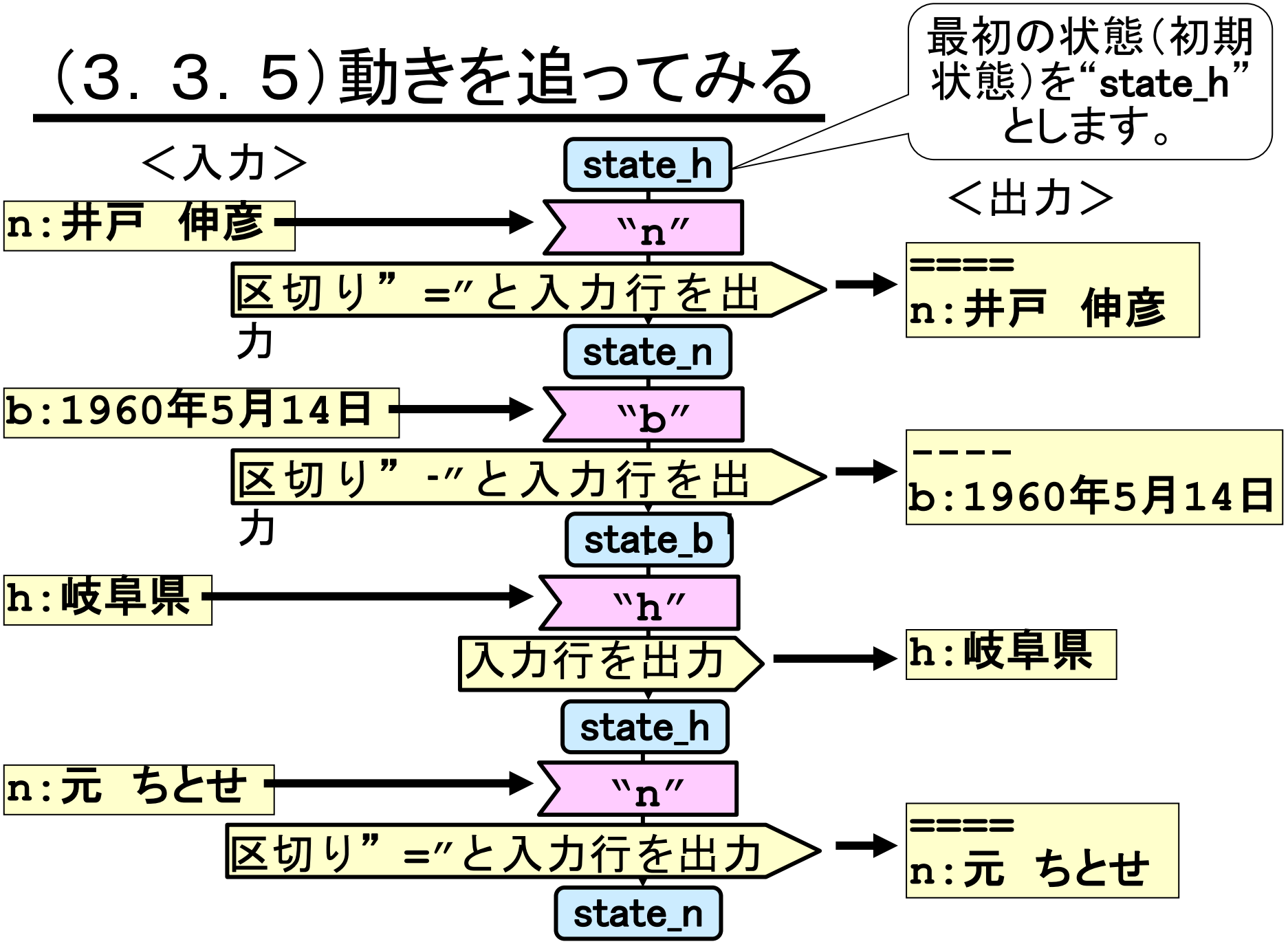


（3．3．4）Judeで作製した状態遷移図

■次のようになります。



（3．3．5）動きを追ってみる



(3．3．6)プログラム　－1／3－

```
import java.io.BufferedReader;
import java.io.FileReader;
import java.io.IOException;
public class Format{
  static final int STATE_N =0;
  static final int STATE_B =1;
  static final int STATE_H =2;
  static int state=STATE_H;
  public static void main(String args[]) throws
 IOException{
    String fn=args[0];
    BufferedReader br=new BufferedReader(new
 FileReader(fn));
    String ss;
    while((ss=br.readLine())!=null){
      char event=ss.charAt(0);
```

“STATE_N”と書けば、“0”を意味するようにする。他も同様。

これ(=sate)が“状態”。最初は“STATE_H”。

引数からファイル名(fn)を読み取る。

“br”からファイルの行を読み取れるようにする。

行がある限り、これを“s”に代入して繰り返す。

最初の１文字をイベントとする。

(3．3．7)プログラム　－2／3－

```
if(state==STATE_N){
  if(event=='n'){
    System.out.println("====");
    System.out.println(ss);
    state=STATE_N;
  }else if(event=='b'){
    System.out.println("----");
    System.out.println(ss);
    state=STATE_B;
  }else if(event=='h'){
    System.out.println("----");
    System.out.println(ss);
    state=STATE_H;
  }
```

状態が“STATE_N”の場合

入力が“n”の場合

区切り(=)を出力

入力行を出力

状態を“STATE_N”とする

他のイベントの場合も、同様に状態遷移する。

(3．3．8)プログラム　－3／3－

```
}else if(state==STATE_B){
        if(event=='n'){
          System.out.println("====");
          System.out.println(ss);
          state=STATE_N;
        }else if(event=='b'){
          System.out.println("ERROR");
          state=STATE_B;
        }else if(event=='h'){
          System.out.println(ss);
          state=STATE_H;
        }
      }else if(state==STATE_H){
        if(event=='n'){
          System.out.println("====");
          System.out.println(ss);
          state=STATE_N;
        }else if(event=='b'){
          System.out.println("ERROR");
          state=STATE_B;
        }else if(event=='h'){
          System.out.println("ERR");
          state=STATE_H;
        }
      }
    }
  }
```

他の状態の場合も、
同様に状態遷移する。

（３．３．９）動かして見る

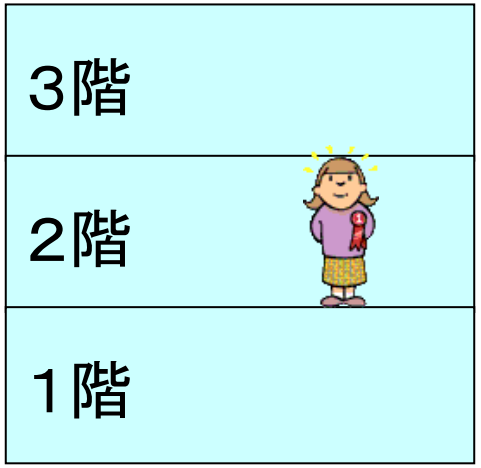
■フォーマットのプログラムを、井戸のネットワークドライブからダウンロードして、動かしてみます。

- 井戸のネットワークドライブから、“マイドキュメント”の下の“javawork”のフォルダに、次の２つのファイルをコピーしてください。
  - **Format.java、test_format.txt**
- eclipseへインポートして実行します。次のスライドを参照してください。
  - 月に吠える　－eclipseによるJavaアプリケーション作成－　スライド（６）ファイルのインポート
- Formatクラスを実行する際の、入力パラメータにて、“test_format.txt”を指定します。

（３．４．１）演習（課題３－１）

■次の状態遷移図をJudeにて作成してください。

- ３階建てのビルを考えます。
- プログラムが起動されると、最初人は１階にいます。
- “u”(up)と入力されると、上の階に移動します。
- “d”（down）と入力されると、下の階に移動します。
- 階を移動した後、今どの階にいるかを出力します。
- １階（３階）に居るときに、“d”(“u”)の入力があった場合は、移動しません。この場合は、移動できない旨を出力します。
- “q”と入力されると、プログラムは終了です。



（３．４．２）演習（課題３－２）

■次のようなプログラムの状態遷移図をJudeにて作成してください。

- 左上側のようなファイルを読み込んで、標準出力に左下側のような出力を行う。
- ファイル中の一人分の記録は、次の通り。
  - “n”が最初に現れる。
  - ”b”と”c” とは、どのような順序でも良い。
  - ”b”は一回、必ず現れる。
  - ”c”は、0回を含めて、何回現れても良い。
  - 順番が狂っていることを発見した場合、プログラムはエラーを出力し、後の動作は保証しない。
- 区切りの記号は、右表の通り。

前	後	区切り
	n	====
n	b	----
n	c	----

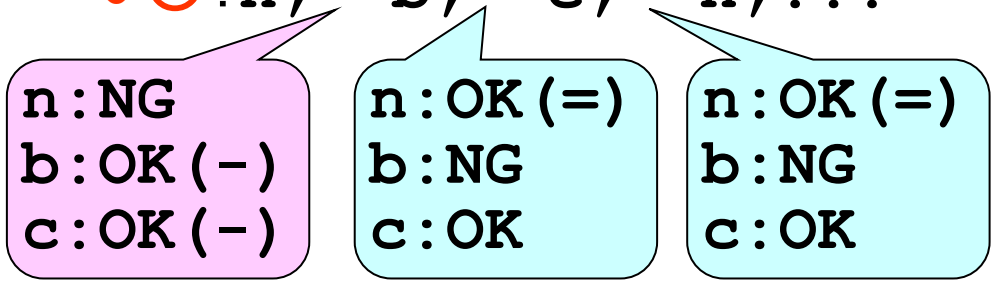
```
n:井戸　伸彦
c:私は踊る。
b:1960年5月14日
c:踊りはサンバ。
n:元　ちとせ
b:1979年1月5日
```

```
====
n:井戸　伸彦
----
c:私は踊る。
b:1960年5月14日
c:踊りはサンバ。
====
n:元　ちとせ
----
b:1979年1月5日
```

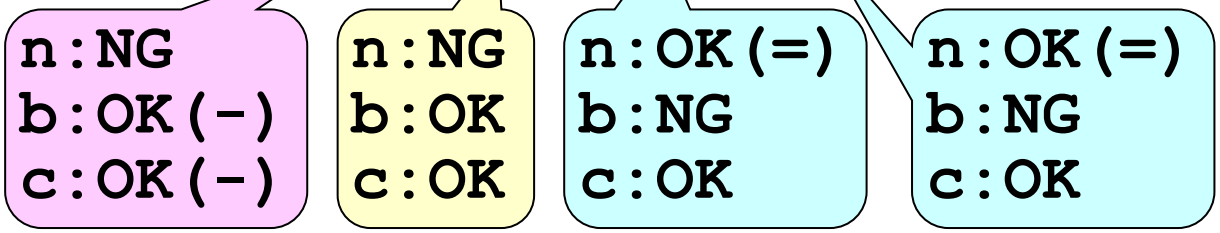
（3．4．3）ヒント1：課題3－2

■"n"、"b"、"c"の現れる例について見ていきます。

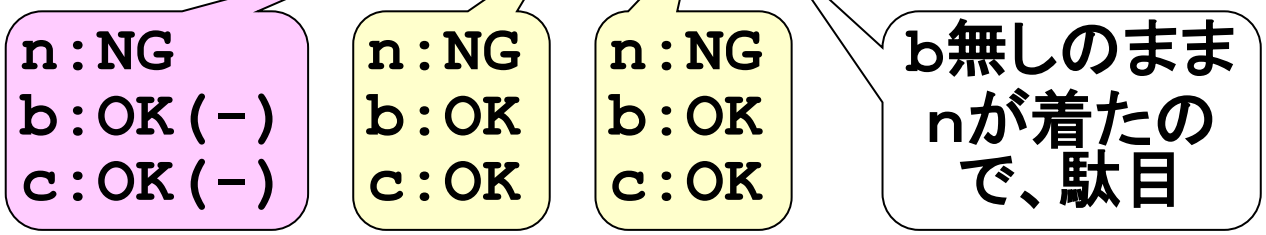
- ○:n, b, c, n,...



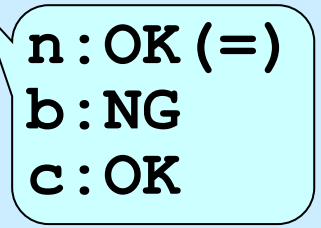
- ○:n, c, b, c, n,...



- ×:n, c, c, n,...



吹き出しは、そのときの入力への対処方法を示します。例えば、



は、
・nが入力ならば、OKで、"="の区切りを入れる、
・bが入力ならば、NGでエラーとなる、
・cが入力ならば、OKで、区切りは入れない、
ことを示します。

（３．４．４）ヒント２：課題３－２

■状態は、次の3つあると考えられます。

■1つの状態は、「次に起こるあらゆるイベントに対して、アクションが同じ場合の集合」と考えることが出来ます。すなわち、前のスライドの吹き出しに示した対処方法の内容が、状態を定義するものと考えられます。

■それぞれの状態の特徴は、次のようにまとめられます。

n:NG b:OK(-) c:OK(-)	n:OK(=) b:NG c:OK	n:NG b:OK c:OK
nを入力した直後	nを入力した直後でなく、nの入力後にbを入力したことがある	nを入力した直後でなく、nの入力後にbを入力したことがない

（３．４．５）演習（課題３－３，４）

■課題３－３

- 課題３－１のプログラムを作成してください。
- 右のようなイメージで動きます。

■課題３－４

- 課題３－２のプログラムを作成してください。
- スライド(3.3.9)のような動きをします。

```
>java kadai33
u
I am on the floor 2.
u
I am on the floor 3.
u
can not go up.
I am on the floor 3
q
>
```

（３．４．６）演習（課題３－５）

■課題３－５

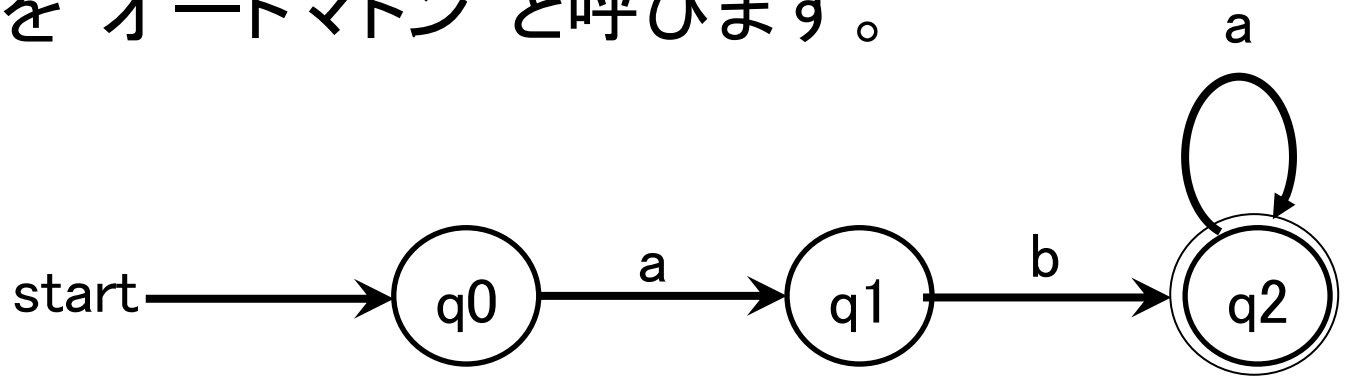
- 次の仕様を満たすプログラムを、状態遷移図を基に作成してください。
  - ◆文字列を入力して、次の分類の順に並んでいることを確認する（どの分類も一文字以上あることとする）。
    - ロアルファベット小文字
    - ロ数字
    - ロ算術演算子の記号（+−*/のみ）
    - ロアルファベット大文字
  - ◆動作例
    - ロ入力：acdd345+/BBYE　出力：OK
    - ロ入力：acdd35t+/A　出力：NG
    - ロ入力：ac33+　出力：NG

※“正規表現とオートマトン”という考え方に続いて行きます。

（３．４．７：参考）正規表現とオートマトン

■最初に‘a’、その次に‘b’、その後に、0個以上の‘a’が続くような並びを、
正規表現では次の“aba*”と表します。

■ある並びが、このような正規表現に沿うものか否かを判定する際、次のような状態遷移のactionが無いようなもの（数理モデル）を使うことが出来ます。
これを“オートマトン”と呼びます。



https://zenn.dev/yumemi_inc/articles/usami_automaton

(3．4．8)演習(課題3－6)

■課題3－1、3－3と同様に、次の状態遷移図とそれに基づくプログラムを作成してください。

- コンロに置いた鍋の中は、最初水の状態で、これを冷却中とします。
- コンロに点火(プログラムでは文字“f”)すると、“カチッ”と音を出し、加熱中になります。
- 加熱中にコンロを消火(プログラムでは文字“e”)すると、“シュ”と音を出し、冷却中になります。
- 上記以外の状態/入力では何もしません。
- なお、プログラムでの状態は、次のように定数で定義してください。

```
static final int COOLING =0;
static final int HEATING =1;
```

（３．４．９）演習（課題３－７）

■課題３－１、３－３と同様に、次の状態遷移図とそれに基づくプログラムを作成してください。

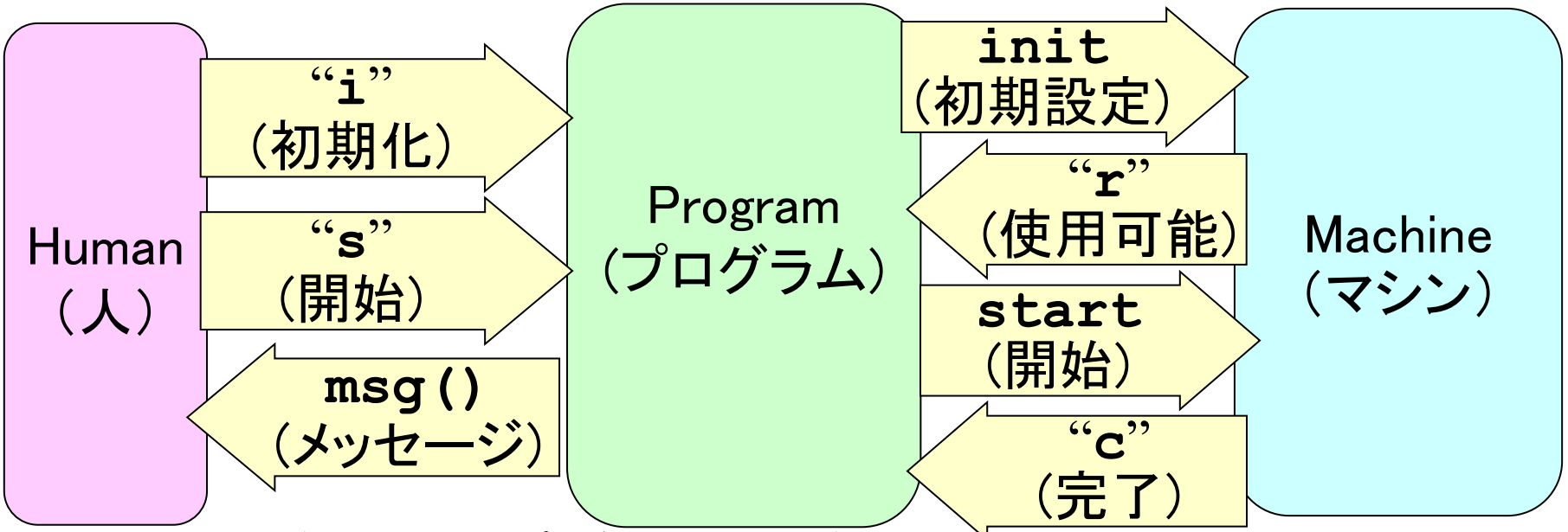
- 鳥が木に止まっています。これを休憩中とします。
- 休憩中に鉄砲の音 (プログラムでは文字“g”)すると、“ギー”と鳴いて、飛行中になります。
- 飛行中に昆虫を見つける（プログラムでは文字“f”）と、“ク―”と鳴いて、追跡中になります。
- 追跡中に昆虫を捕まえる（プログラムでは文字“c”）と、“グエ”と鳴いて、飛行中になります。
- 飛行中に木を見つける（プログラムでは文字“t”）と、“ピー”と鳴いて、休憩中になります。
- 上記以外の状態/入力では何もしません。

```
static final int REST =0;
static final int CHASE =1;
static final int FLIGHT =2;
```

（4．1．1）マシン制御

■次のようなプログラムを考えます。

- プログラムは、Human(人)とMachine(マシン)との間に立って、制御を行います。
- 人⇔プログラム、プログラム⇔マシンの入出力には、次のようなものがあります。
- マシンの動きについては、次のスライド(4.1.2)で説明します。



（４．１．２）マシンの仕様（使い方）

■Machineは、電源ON時には使用できず、初期設定(**init**)を発行する必要があります。

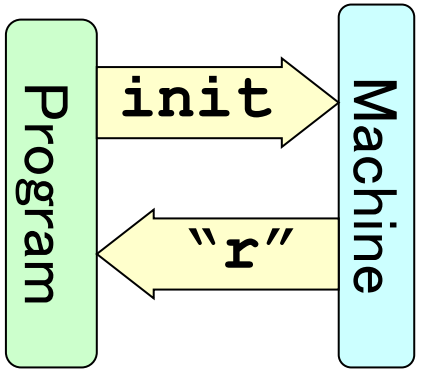
- 初期設定(**init**)を発行した後、準備完了となると、マシンは、文字“**r**”をProgramに送ってきます。これ以降、使用が可能となります。

■使用可能となってから、開始（**start**）を発行すると、マシンは仕事をし始めます。仕事が終わると、文字“**c**”をProgramに送ってきます。

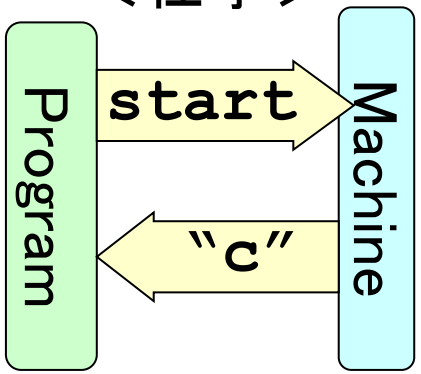
- “**c**”を受け取る前に、再度開始(**start**)することは出来ません。
- “**c**”を受け取った後は、次の開始（**start**）を発行することが出来ます。

■マシンが調子が悪くなった場合、初期設定(**init**)を発行すると直る場合があります。

＜初期設定＞



＜仕事＞



（４．１．３）プログラムの仕様

■次の点を前提として、プログラムの仕様を考えて見ます。

- Human（人）とのインタフェースは、スライド（４．１．１）に示した3つのみとする。
- タイマーは使わないこととする（実際には、このような場合に使うことが普通です）。

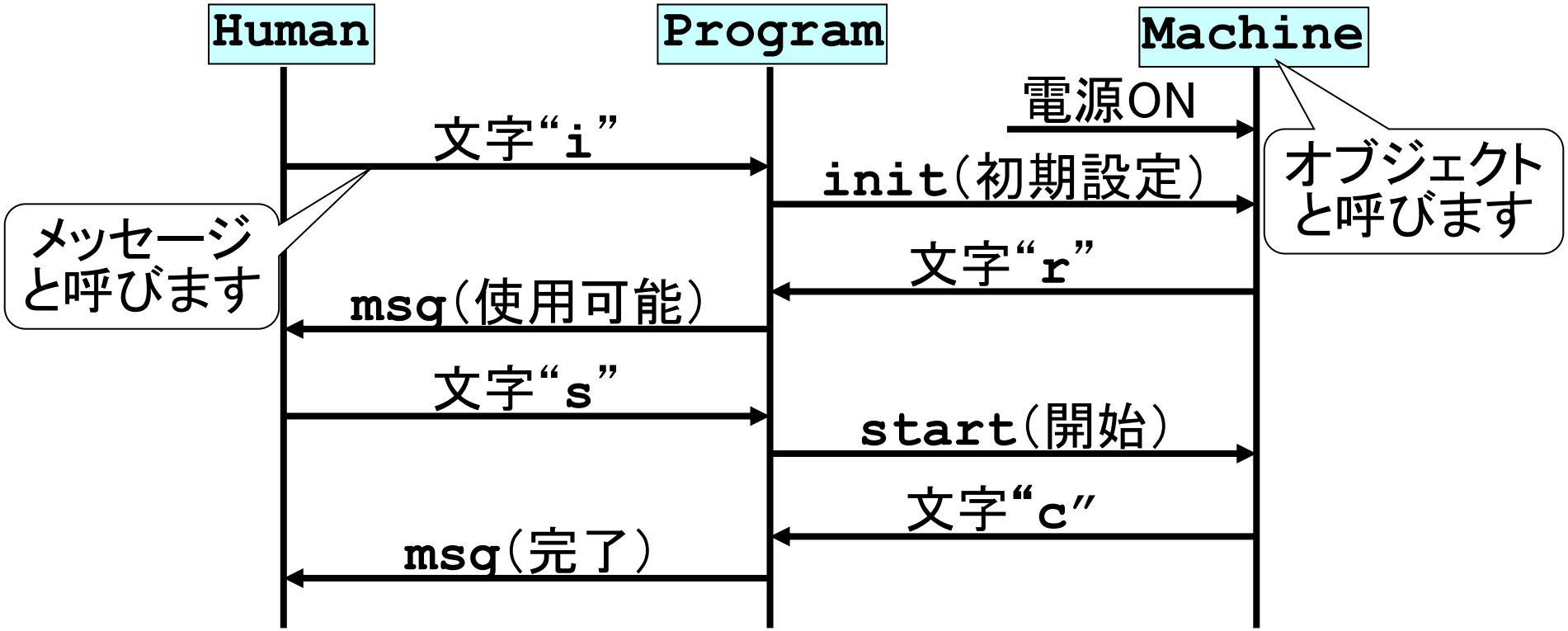
■次のような方針で考えます。

- Human（人）からのコマンドのうち、Machine（マシン）の動作状態にそぐわないものは、**msg**（メッセージ）を発行して、拒絶する。
- 但し、Machine(マシン)からの応答が無いことがありえる状態では、必ず文字“**i**”(初期化)を受け入れて、**init**（初期設定）を実行する。
- Machine(マシン)から、想定されているもの以外の入力があった場合は、故障とみなす。

（４．２．１）解りやすい図にする

■スライド(4.1.1)～(4.1.3)では、Human,Program,Machine が具体的にどのように動くかが想像しにくいと思います。

■「図にすると解り良い」というのは、よくあることです。例えば、最初の動きは次のような図に出来ます。



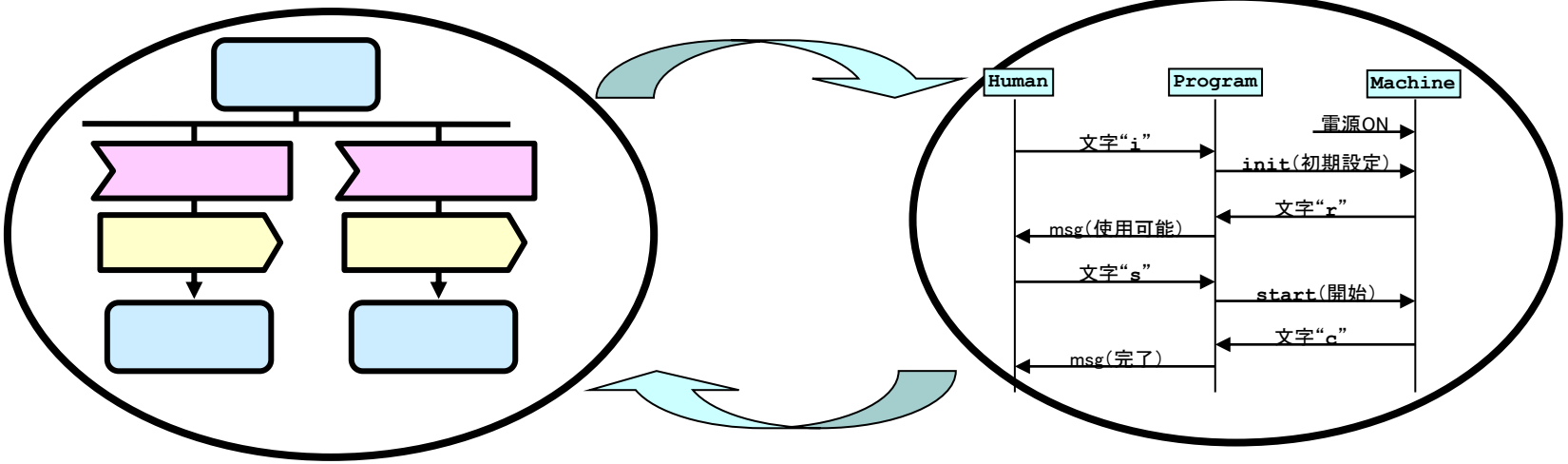
（4．2．2）シーケンス図

■スライド(4.2.1)のような図を、シーケンス図といいます。

■複雑な状態遷移を考えなければならない場合、状態遷移図とシーケンス図とを作成しながら設計を進めていくことが一般的です。一方を考えながら、他方を手直ししていく訳です。

＜状態遷移図＞

＜シーケンス図＞



（4．2．3）状態遷移図との違い

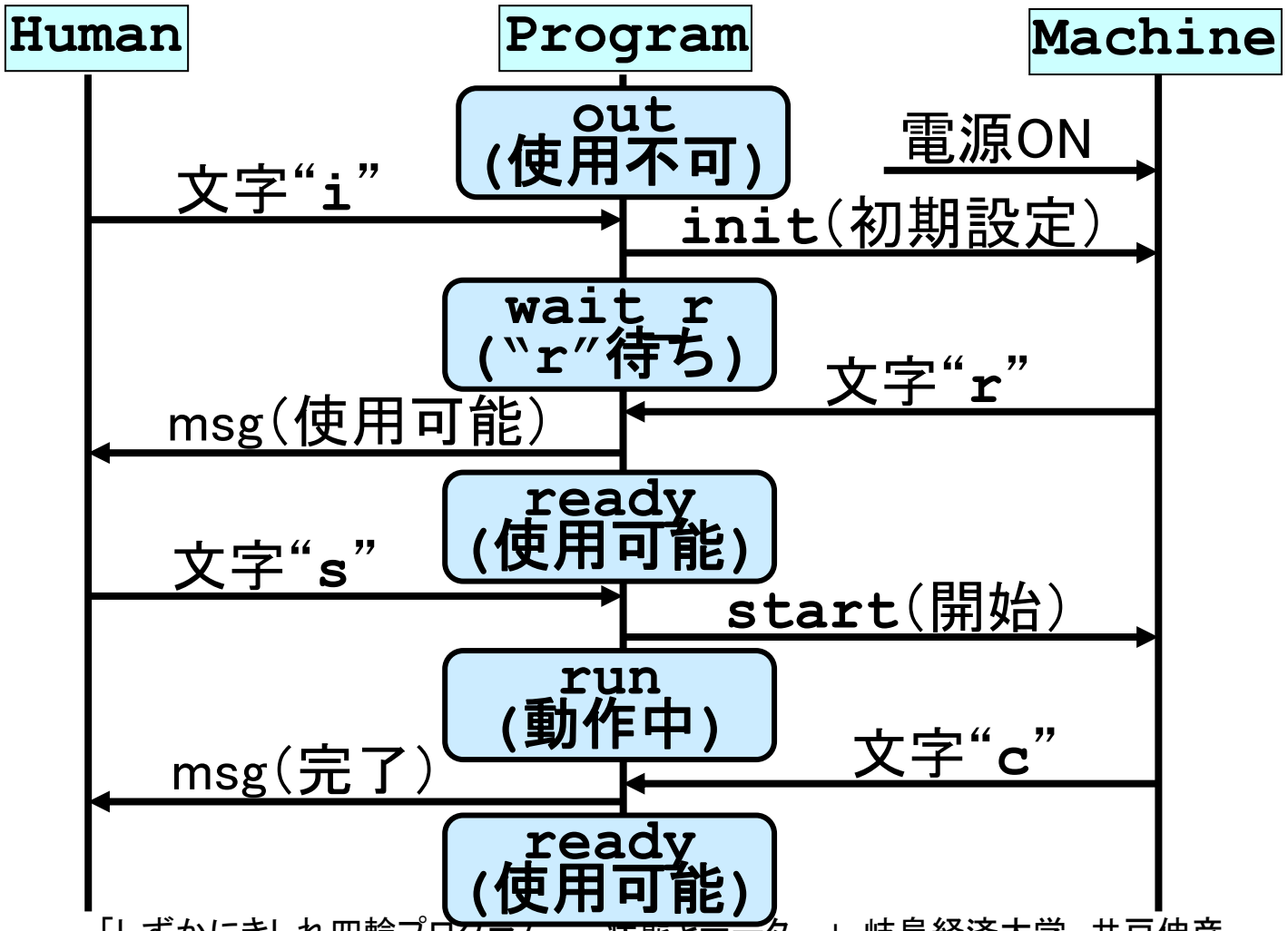
■状態遷移図では、状態を持つひとつの“もの”に対してのみの動きをすべて表します。

■一方、シーケンス図では、複数の要素も書き加えて、相互の動作関係を示します。但し、シーケンス図ではすべての場合ではなく、ある特定の場合についての相互動作関係を示します。

	状態遷移図	シーケンス図
示す対象	1つ	複数
示す場合	すべての場合	ある特定の場合

（4．3）状態を書き加えたシーケンス図

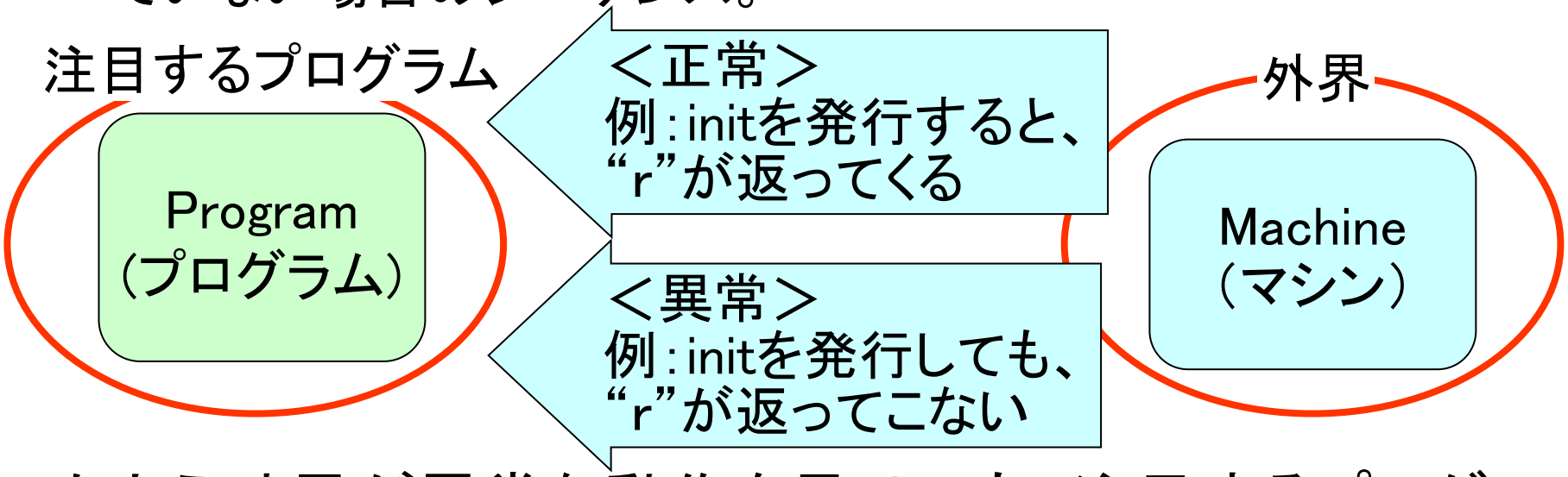
■シーケンス図に状態を書き加えると、状態遷移図を作成していく上で、一層効果的です。



（4．4．1）正常シーケンスと異常シーケンス

■シーケンス図は、次のように分類されます。

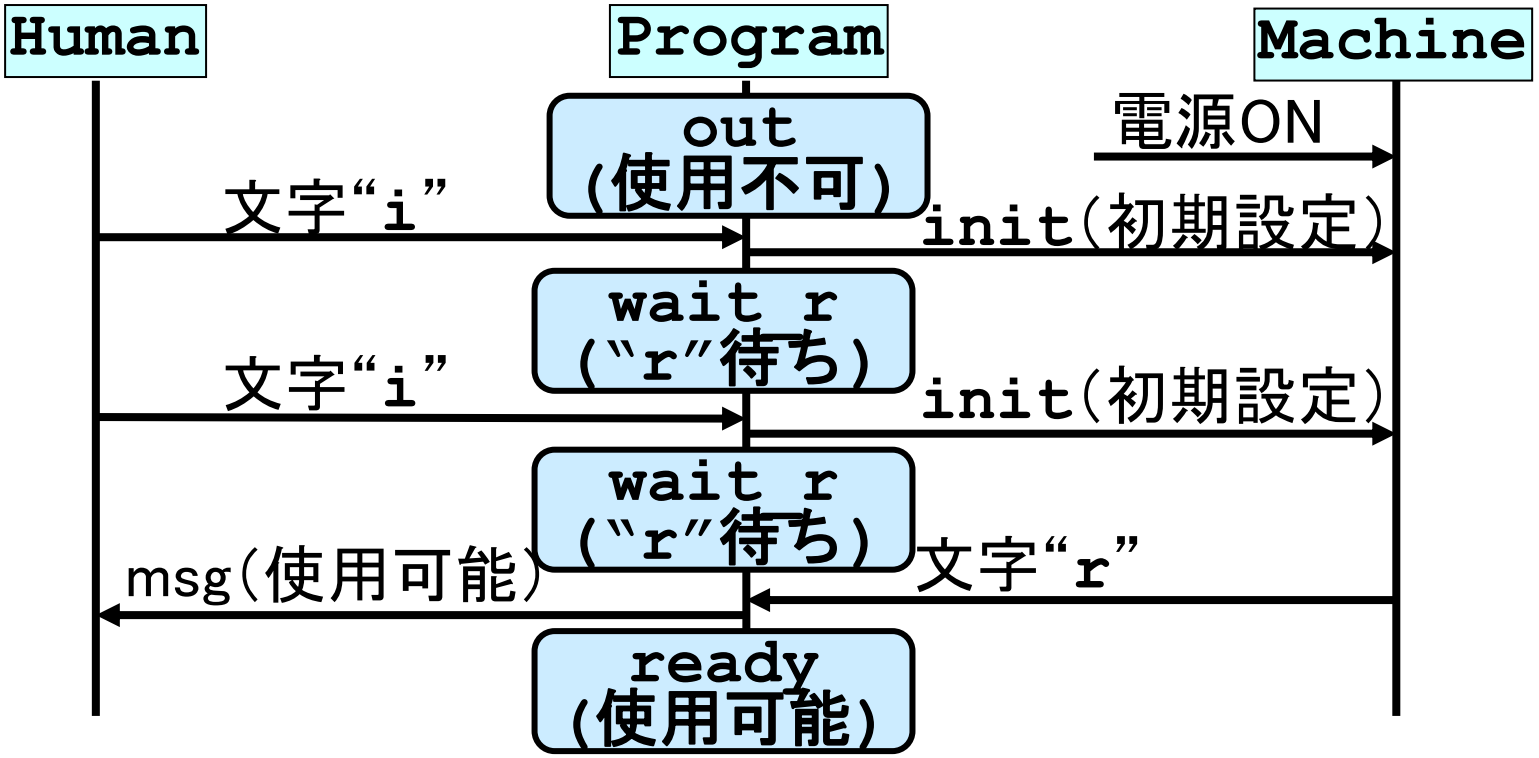
- 正常シーケンス：注目するプログラムの外界が正常に動作している場合のシーケンス（スライド(4.3.1)は正常シーケンス）。
- 異常シーケンス：注目するプログラムの外界が正常に動作していない場合のシーケンス。



■たとえ外界が異常な動作を見せても、注目するプログラムはそれなりの対処をする必要があり、異常シーケンスも考えなくてはなりません。

（４．４．２）応答が無い場合

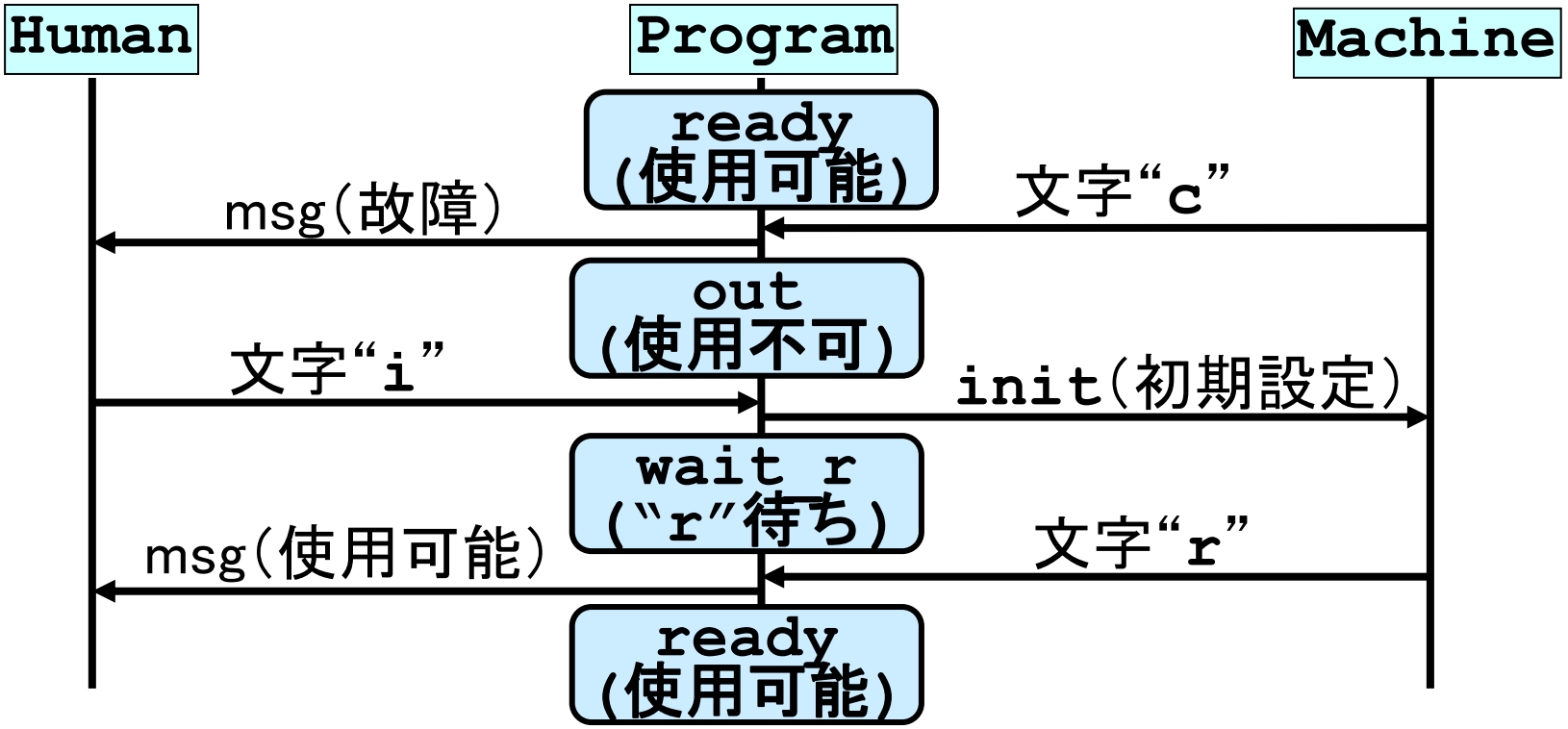
■“init”をMachineに発行して、応答である”r”が返ってこない場合、Humanがこれに対応して、再度文字”i”を投入することにします。



■この異常シーケンス図より、「wait_r(“r”待ち)の状態でも、Humanからの入力である文字”i”は受け付けなければならない」ことが解ります。

（4．4．3）脈絡なく応答を受けた場合

■マシンに対して、”start”を発行していないのに、文字”c”を受け取った場合のシーケンスを示します。

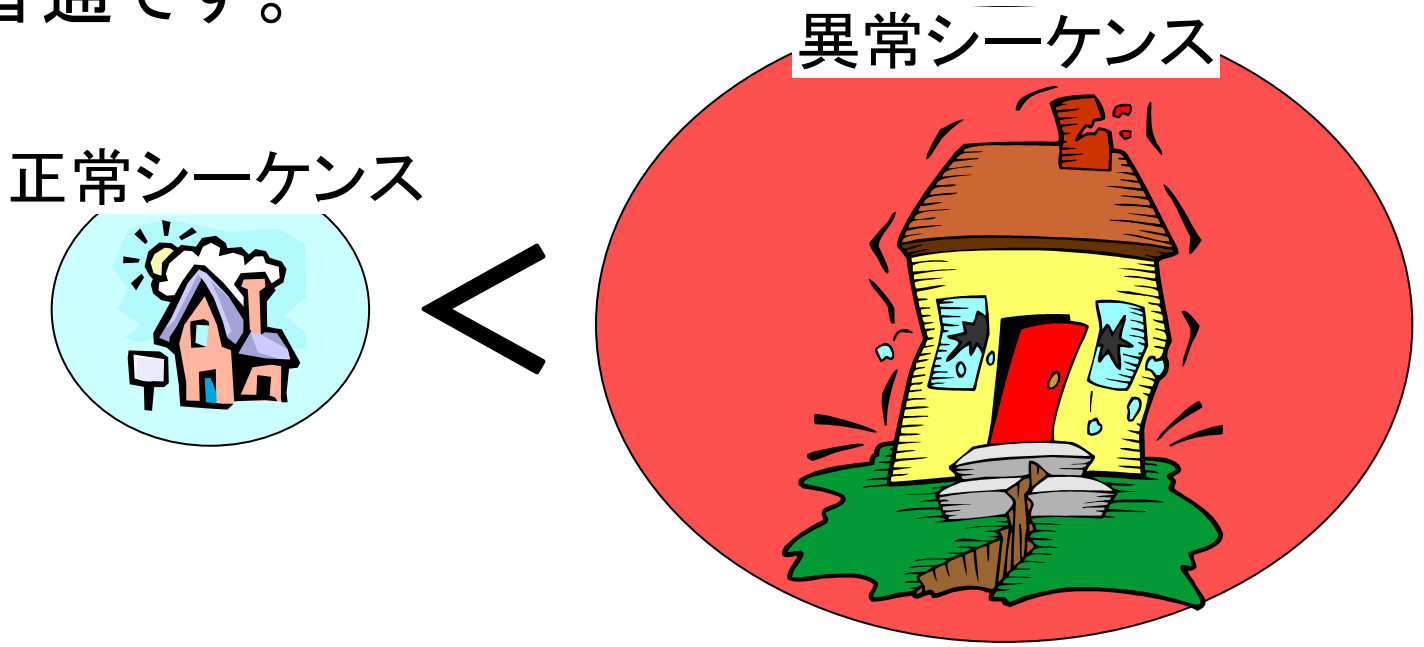


■Humanに故障を通知して、初期設定の文字“i”が入力されることを期待している訳です。脈絡なく受け取った”c”を無視する仕様も可能です。ここでどのように対応するかを決めることが、「仕様を設計する」ことになります。

（4．4．4）異常シーケンスの数

■一般に、正常シーケンスに比べて異常シーケンスの数は多くなることが普通です。外界で起こって欲しくないことは、すべて起こるとして考えねばなりません。

■プログラムを書くときも、正常シーケンスに対応した部分よりも、異常シーケンスに対応した部分が大きくなることが普通です。



（４．４．５）Webアプリケーション

■どのような場合でも、異常シーケンスを熱心に作りこむことが良いという訳ではありません。

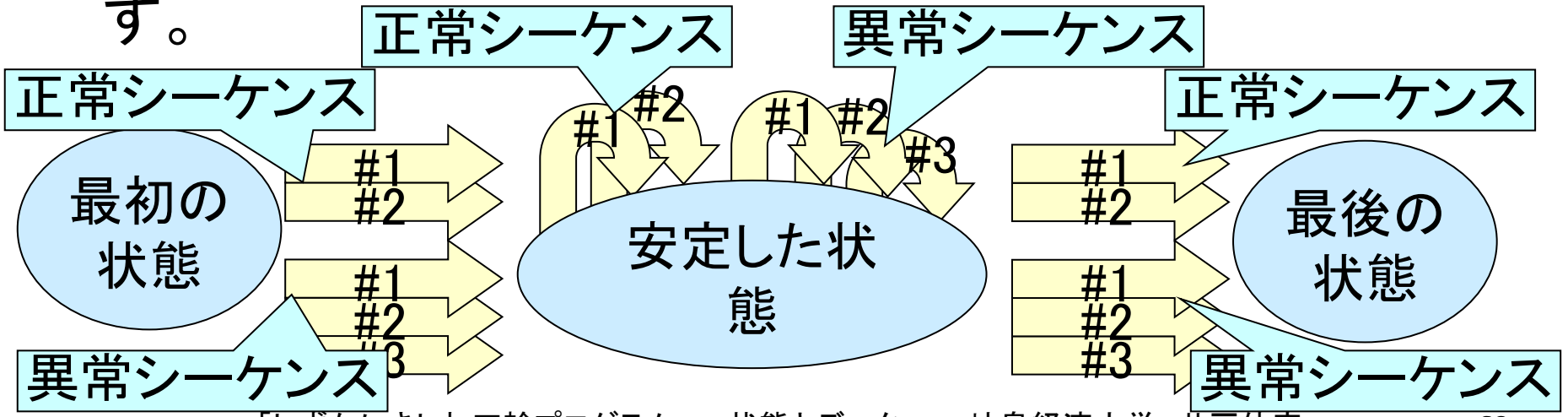
■例えば、Webアプリケーションの場合は、セキュリティを守るために、次のような方針で作るべきです。

- ブラウザとの相互動作は、想定したもの以外は許容しない。
- ブラウザからの入力は、想定したものであることを厳しくチェックする。

■上記のような方針に基づけば、異常シーケンスの数は少なくなります。しかしながら、Webアプリケーションのようなプログラムばかりではないことも確かです。

（４．５．１）シーケンス図を作成する単位

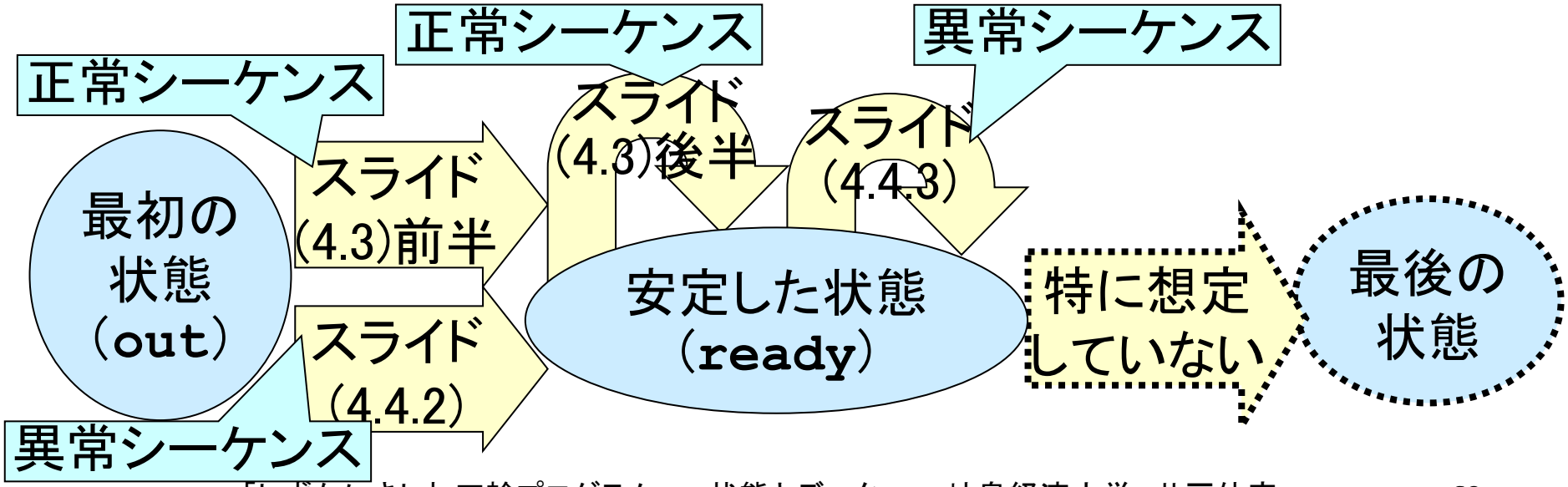
- ■シーケンス図は特定の場合の相互動作関係を示すものですから、あらゆる動きについて作成しようとすると、無限に数が増えてしまいます。
- ■同じような動きのシーケンスを沢山作成することは無意味ですから、意味を持った単位ごとに作成する必要があります。
- ■通常は、「最初の状態」、「安定した状態」、「最後の状態」で区切られた部分を、シーケンス図として作成します。



（４．５．２）マシン制御の場合

■マシン制御の場合、安定した状態は、“**ready**”となります。
- 安定した状態が複数ある場合もあります。
- 安定した状態以外を、“過渡状態”と呼ぶこともあります。
- 最初の状態から、安定した状態までのシーケンスを“初期設定”シーケンスと呼ぶことがあります（最後の状態に至るシーケンスを“終了シーケンス”という言い方もします）。

■スライド(4.3)では、初期設定シーケンスに正常動作のシーケンスが加えられています(すなわち、(4.5.1)の整理は、一応の目安と考えて置いてください)。

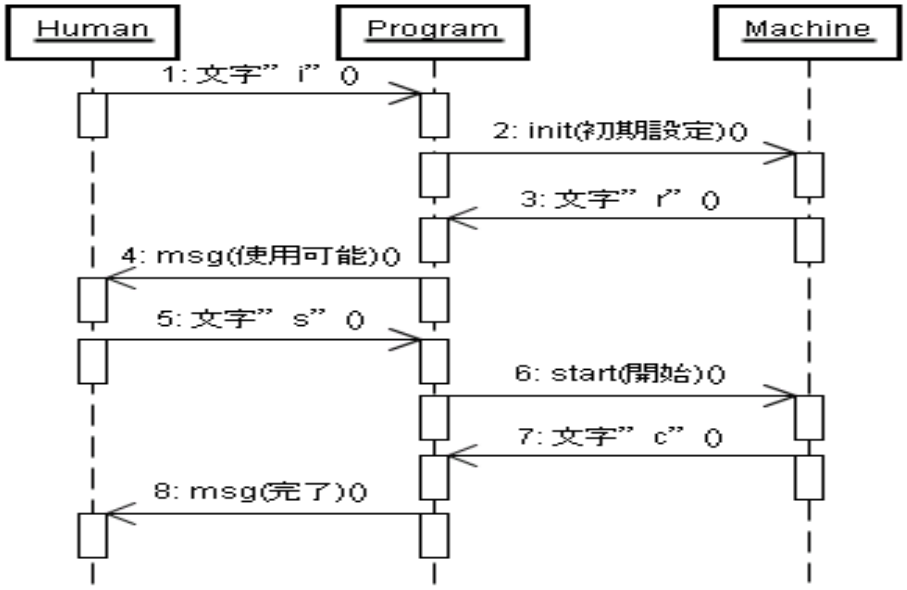


（４．６）Judeにてシーケンス図を描く

■Judeには、スライド(4.2.1)のような状態を加えないシーケンス図を描く機能があります。

■UMLのシーケンス図には状態を書かないので、Judeに状態を描く機能はありません。状態を書き加える場合には、単なるテキストとして書き加えることにします。

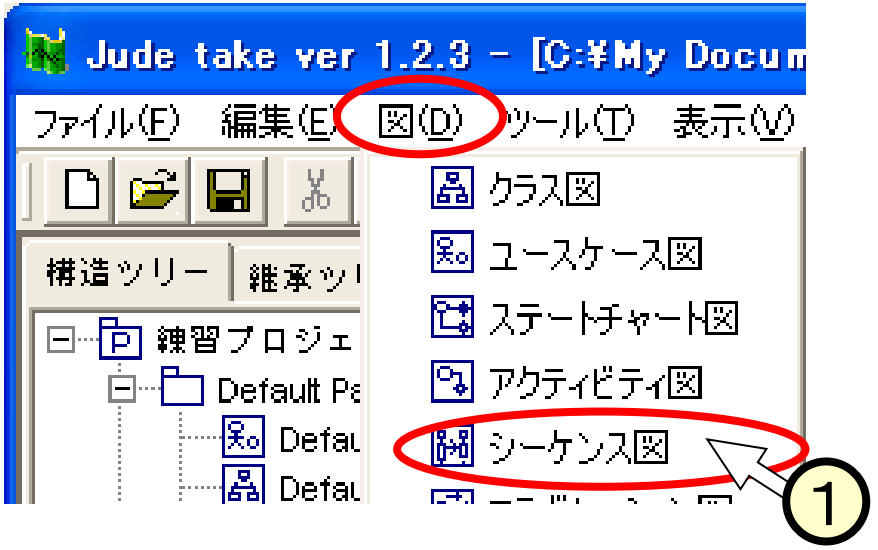
■見栄えとしては、下図のようになります。

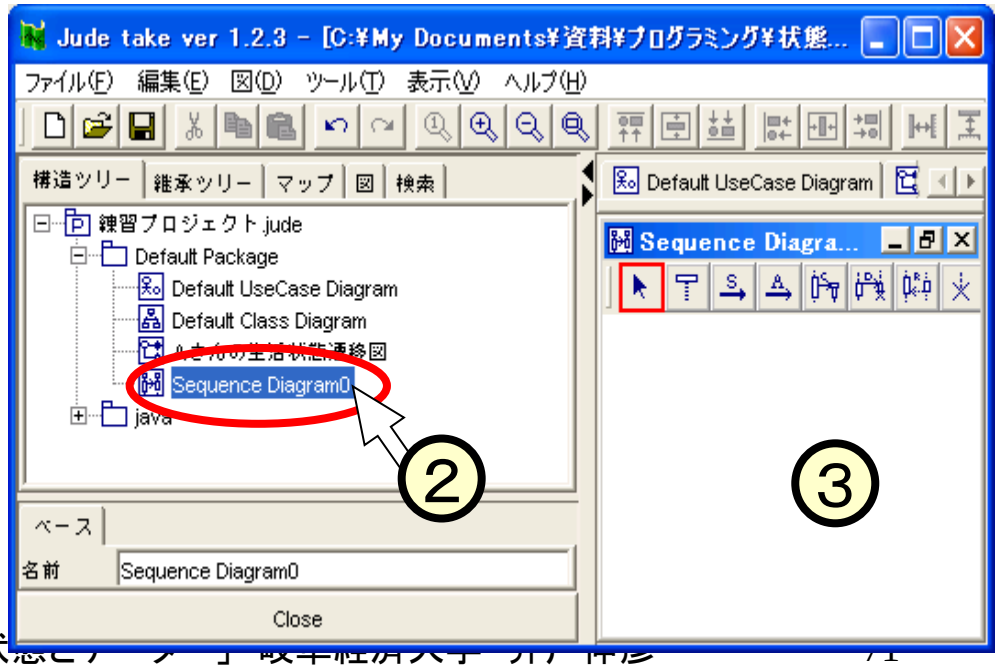


（4．6．1）新規シーケンス図を開く

■[図]–[シーケンス図]をクリック（①）します。

■プロジェクト・ビューに新しい図（Sequence Dialog）が表示され、これをクリック（②）すると、ダイアログ・エディタ（③）にて図の作成が可能になります。





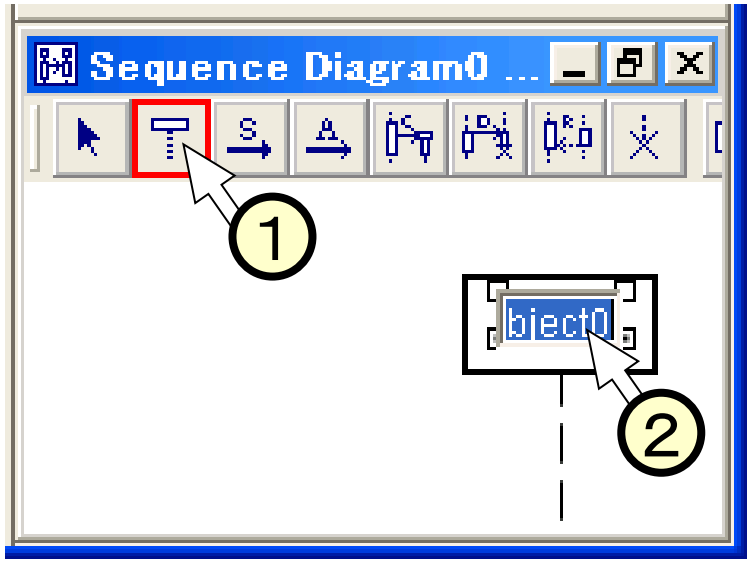
（4．6．2）オブジェクト要素の配置

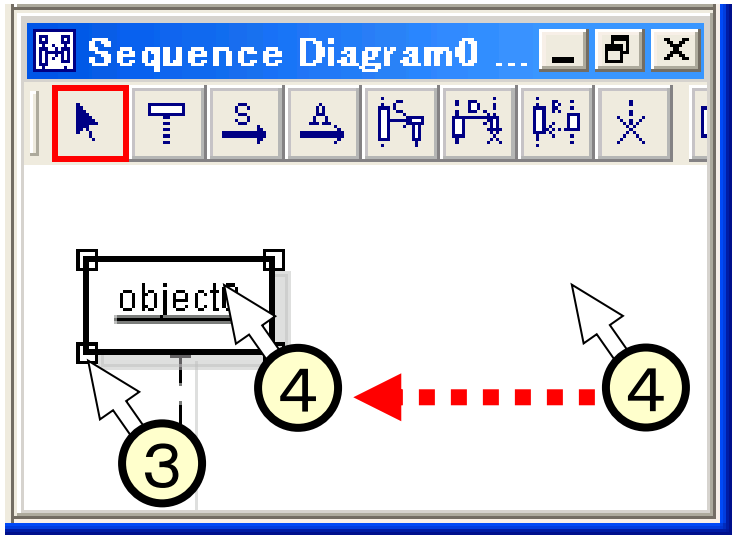
■ツールバーの[オブジェクト]をクリック（①）して、ダイアログ・エディタ上をクリック（②）します。

■オブジェクト要素が、名前の入力待ちの形で配置されます。ここで名前の入力が可能ですが、後から名前の欄をダブルクリックして入力することも出来ます。

■ドラッグ（③）することで、移動させます（上下の位置は動きません）。また、要素の四隅のつまみをドラッグ（④）して、大きさを調整します。

■フォーカスされた状態（＝四隅につまみが表示）で[Delete]キー押下すると、要素は削除されます。



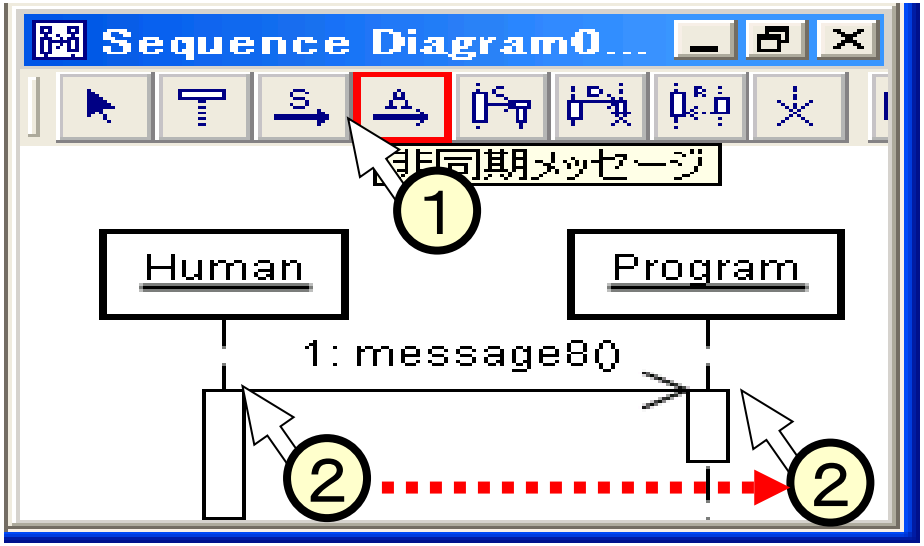


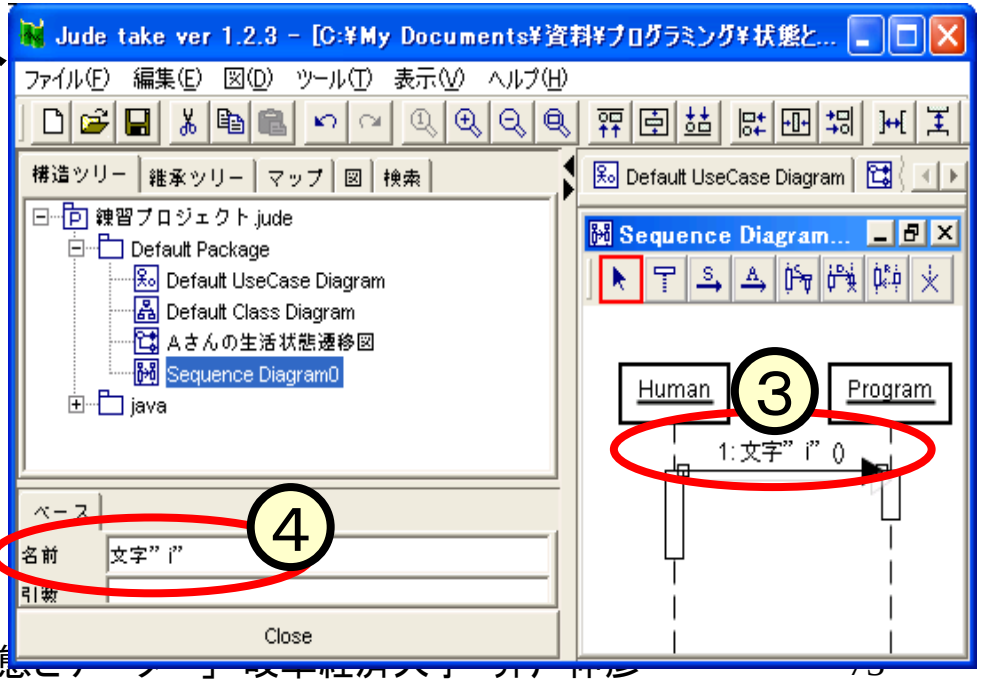
（4．6．3）メッセージ要素の配置

■前スライドのようにして、2つのオブジェクト要素を作ったとします。この2つの間に、メッセージ要素を付け加えます

■ツールバーの非同期メッセージ要素をクリック(①)して、メッセージ送信元のオブジェクトから受信先のオブジェクトへドラッグ(②)します。

■メッセージ名は、ダイアログ・エディタ上(③)、もしくは、メッセージをフォーカスした際のプロパティ・ビュー上(④)で入力できます。



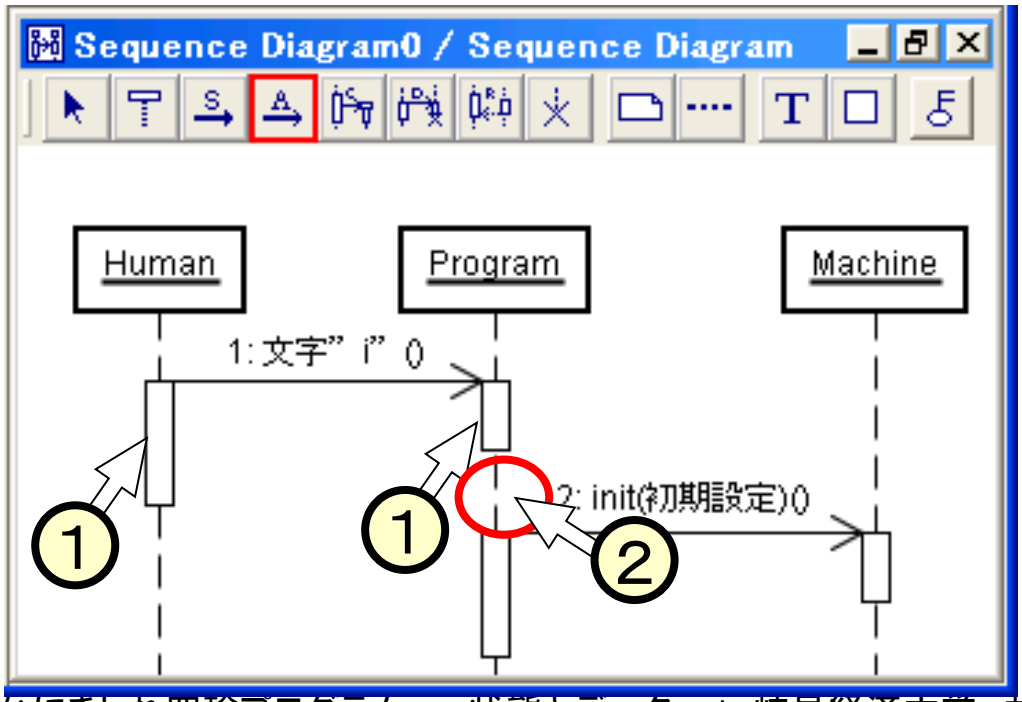


（4．6．4）オブジェクト上の白四角

■メッセージを配置すると、オブジェクトの縦点線上に白四角（①）が描画されます。

■これには意味がありますが、ここでは無視してください。

■2つめ以降のメッセージを配置する際、白四角が重ならないように（②）しておいてください。



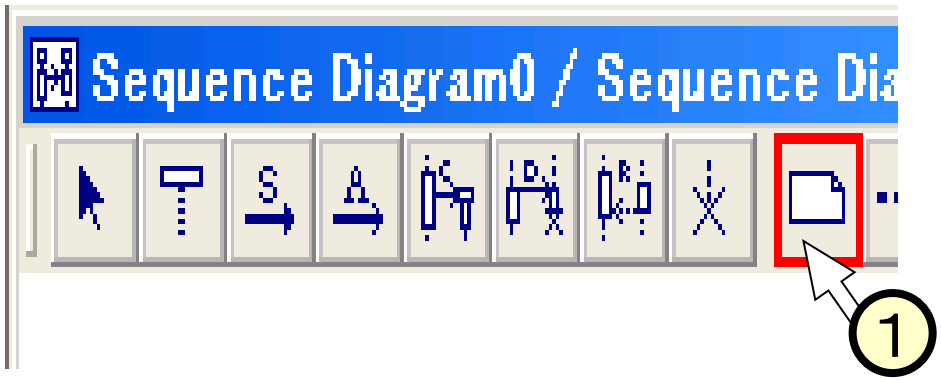
（4．6．5）状態の書き加え

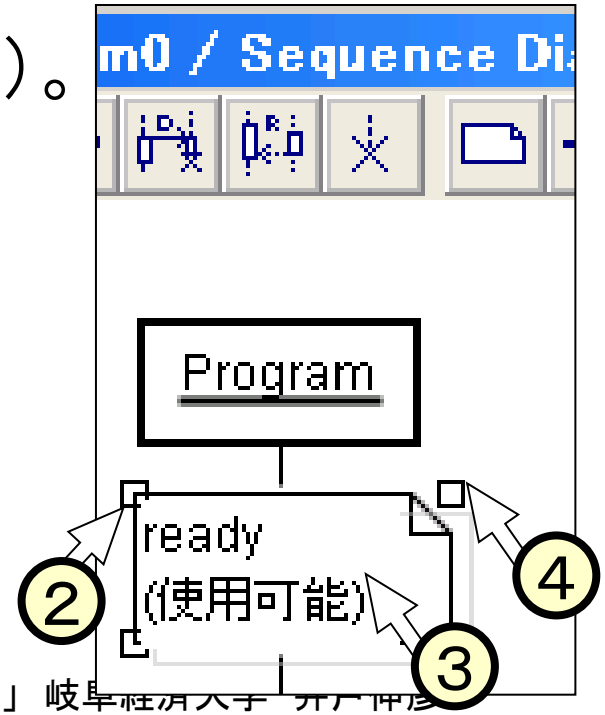
■前述のとおり、Jude（UML）ではシーケンス図に状態は描きません。代替として、“ノート”を使うことにします。

■ツールボックスにて、[ノート]ボタン（①）をクリックします。

■適当な位置をクリック（②）してノートを配置し、状態名を入力（③）します。

■位置／大きさを調整します（④）。

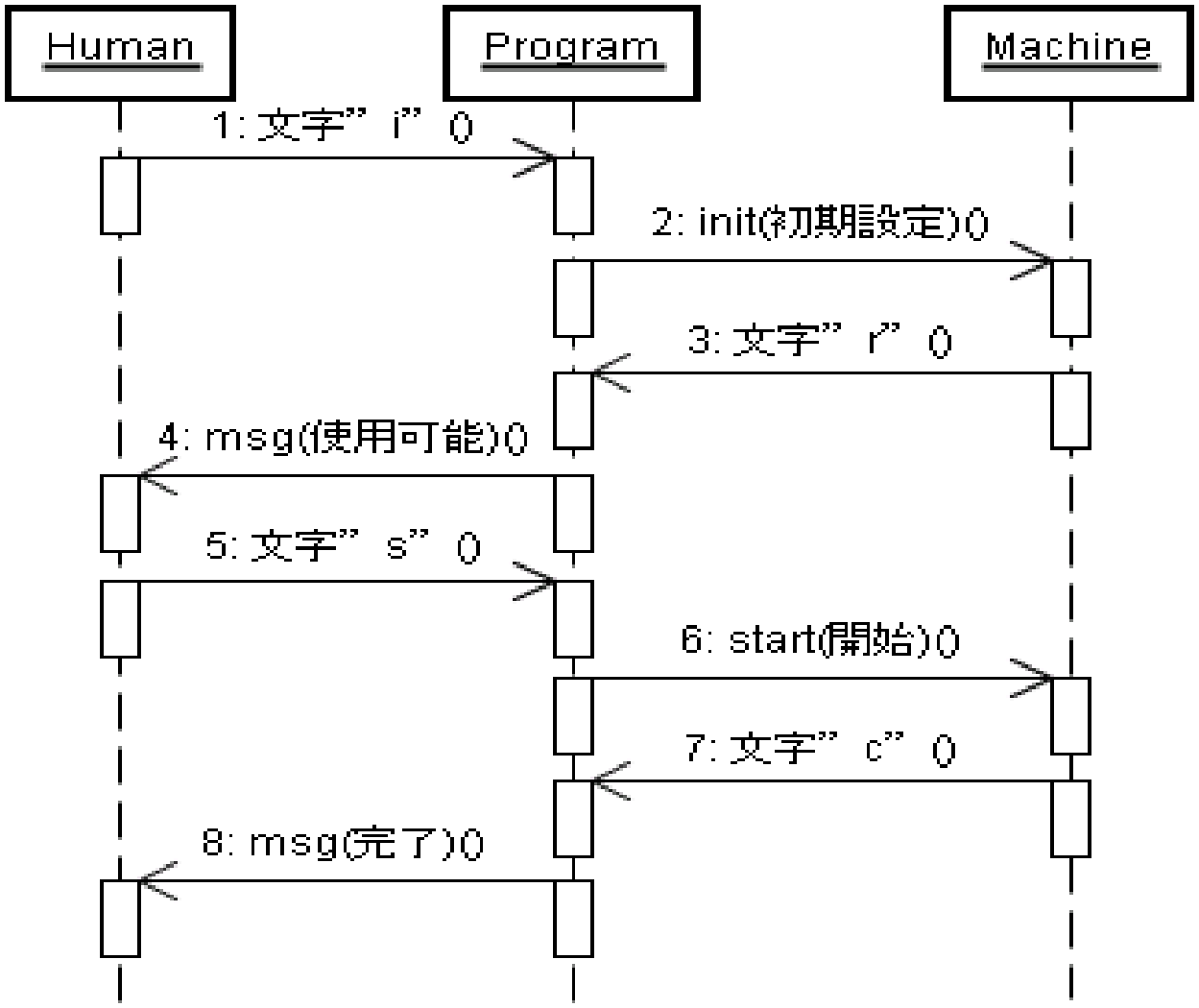




（4．7）演習（課題4－1）

■スライド(4.2.1)のシーケンス図をJudeにて作成してください。

■解答：

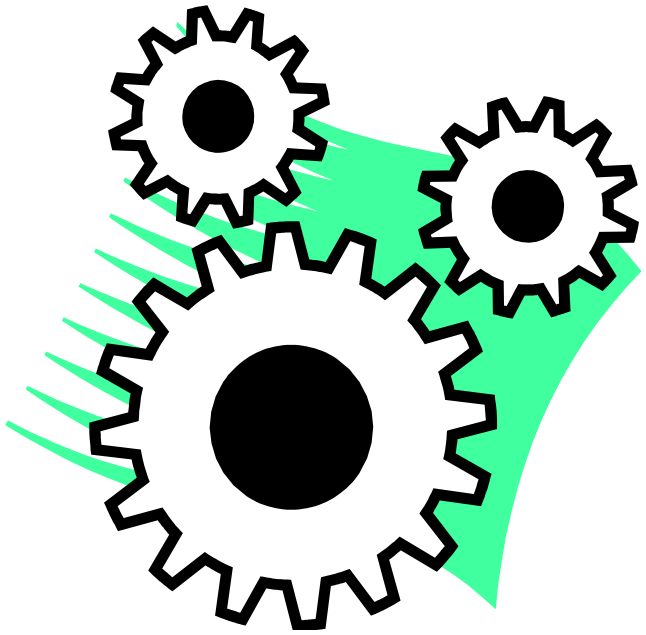


（４．８）演習（課題４－２）

■“**start**”をMachineに発行して、応答である”**c**”が返ってこない場合のシーケンス図を、状態を書き加えてJudeにて作成してください。

■ヒント

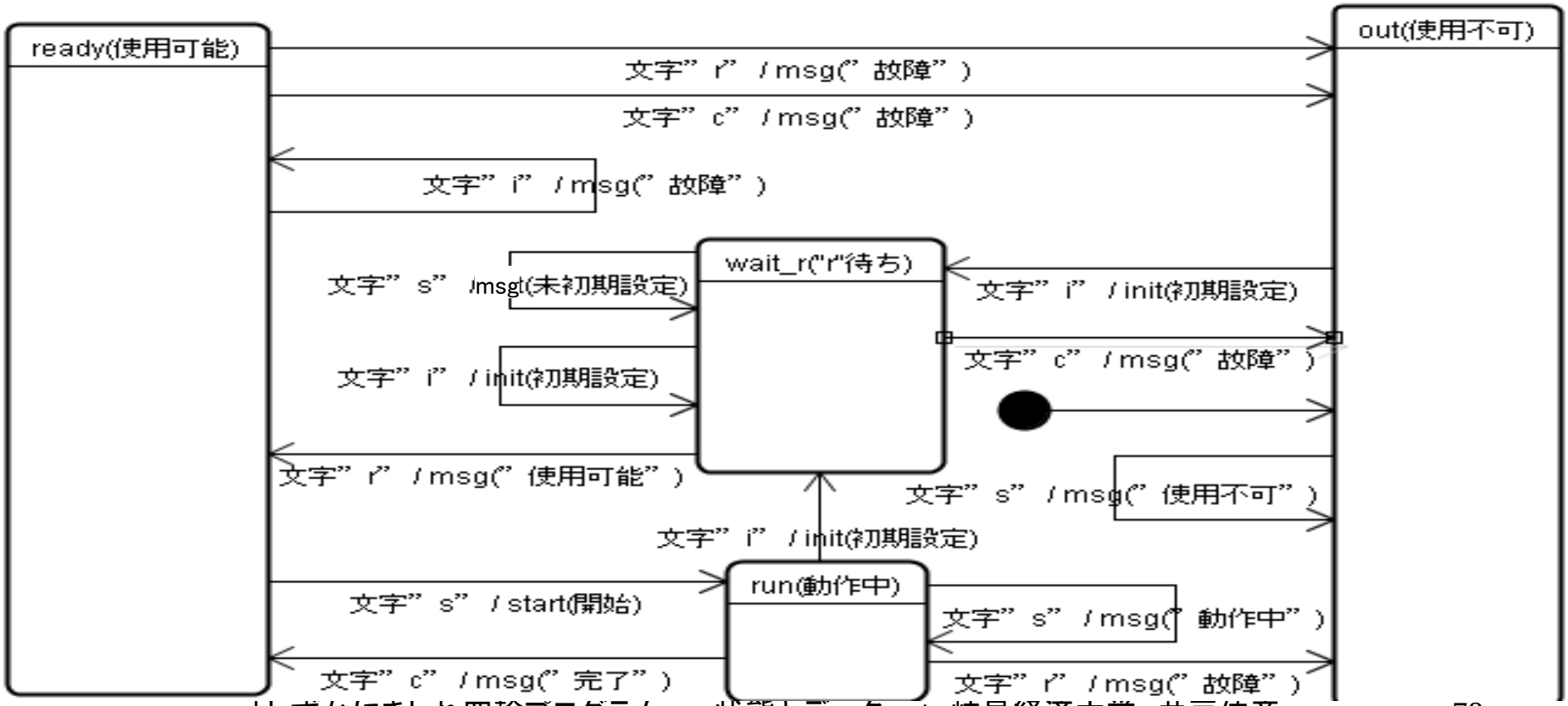
- スライド(4.4.2)と似たシーケンス図になります。



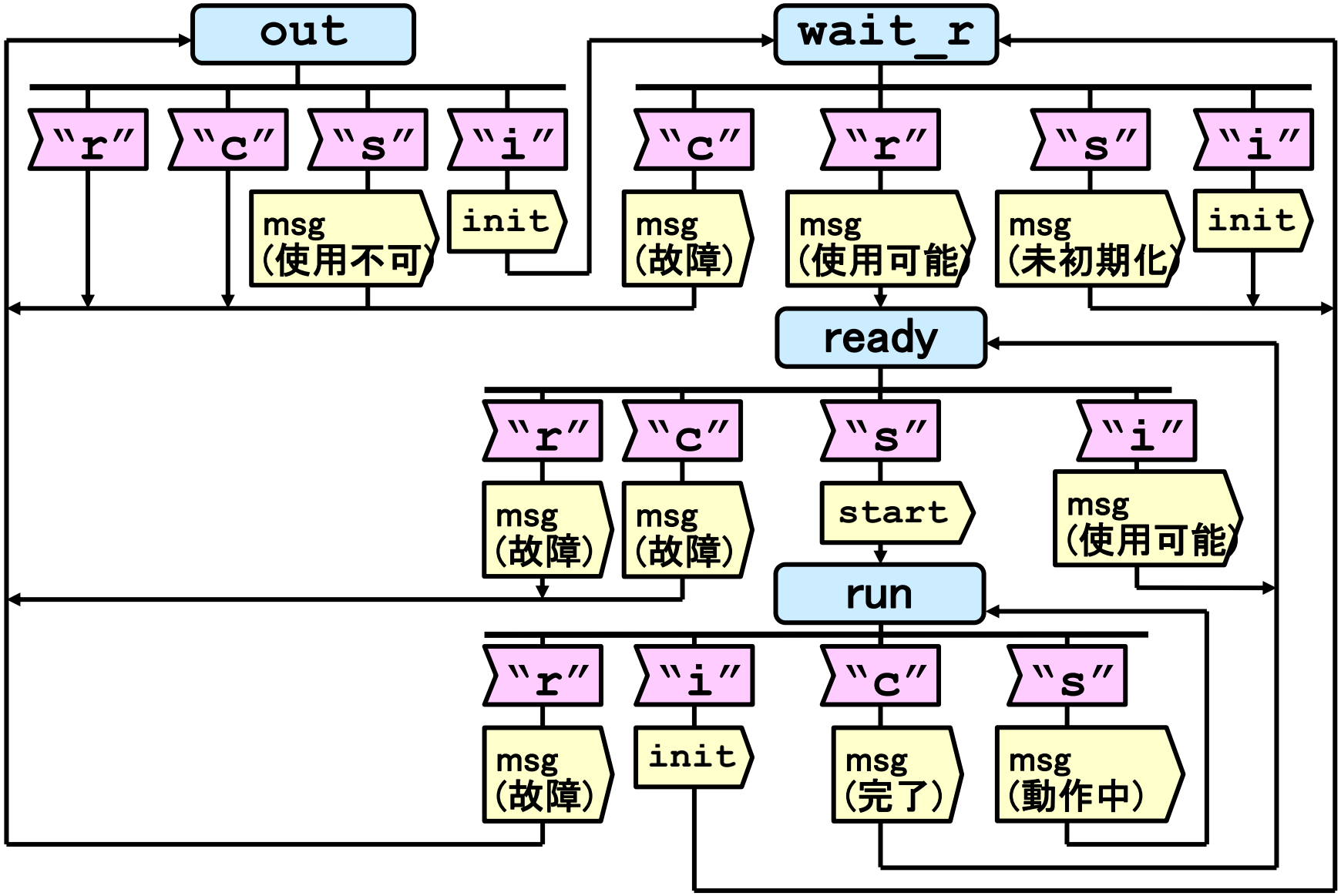
（4．9．1）マシン制御の状態遷移図

■下図のような状態遷移図としました。

- スライド(4.1.3)に記した方針が守られていることを確認してください。黒丸（●）は、開始を示す記号です。
- 印刷が見づらい点は、次スライドのSDL版で補ってください。

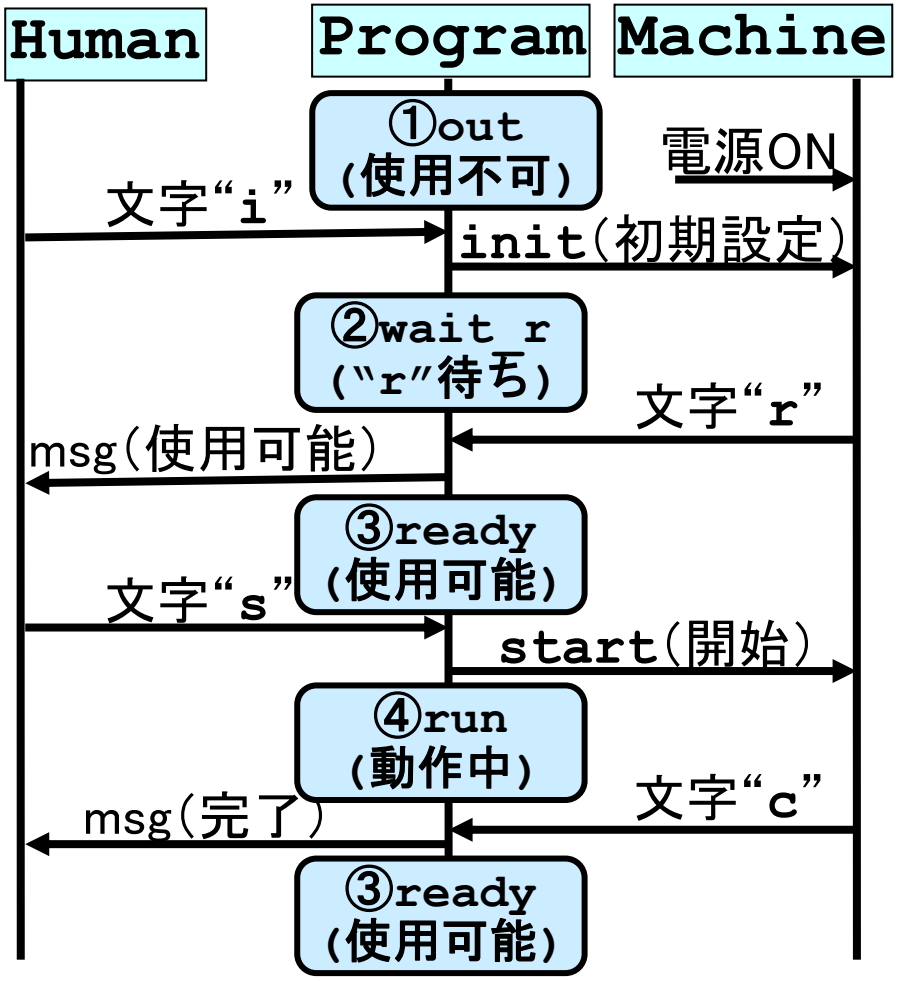


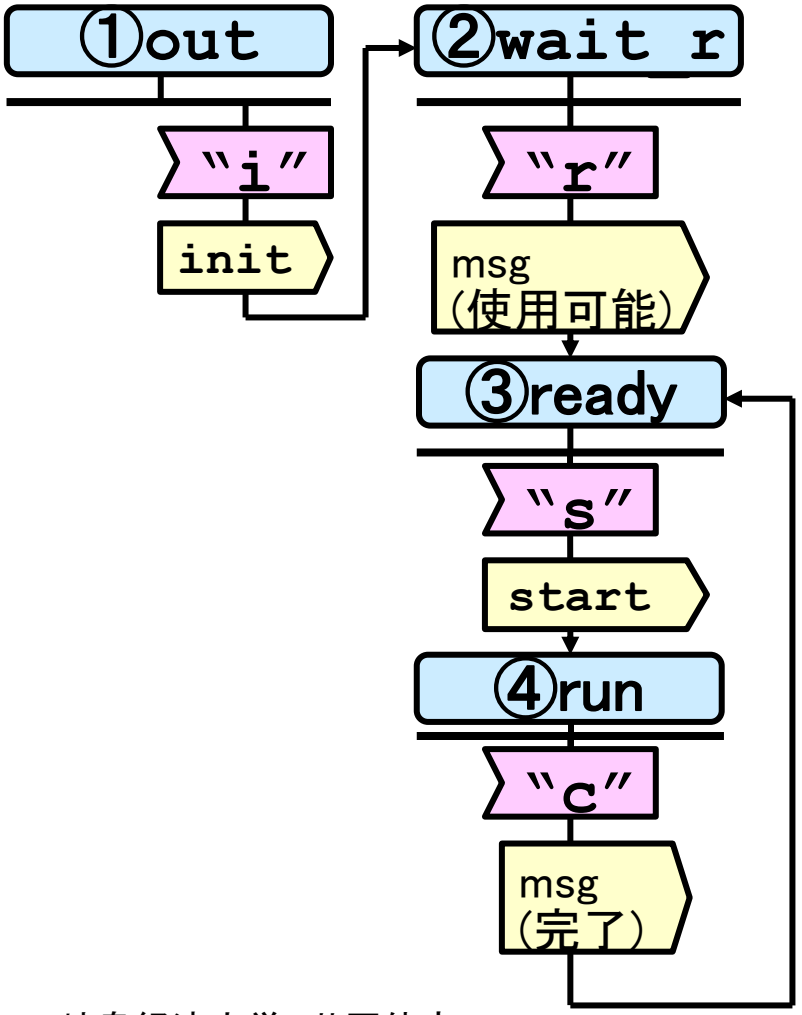
（4．9．2）マシンの状態遷移図（SDL）



（4．9．3）シーケンス図と状態遷移図

■シーケンス図は、状態遷移図の一部と正確に対応していることがわかります。

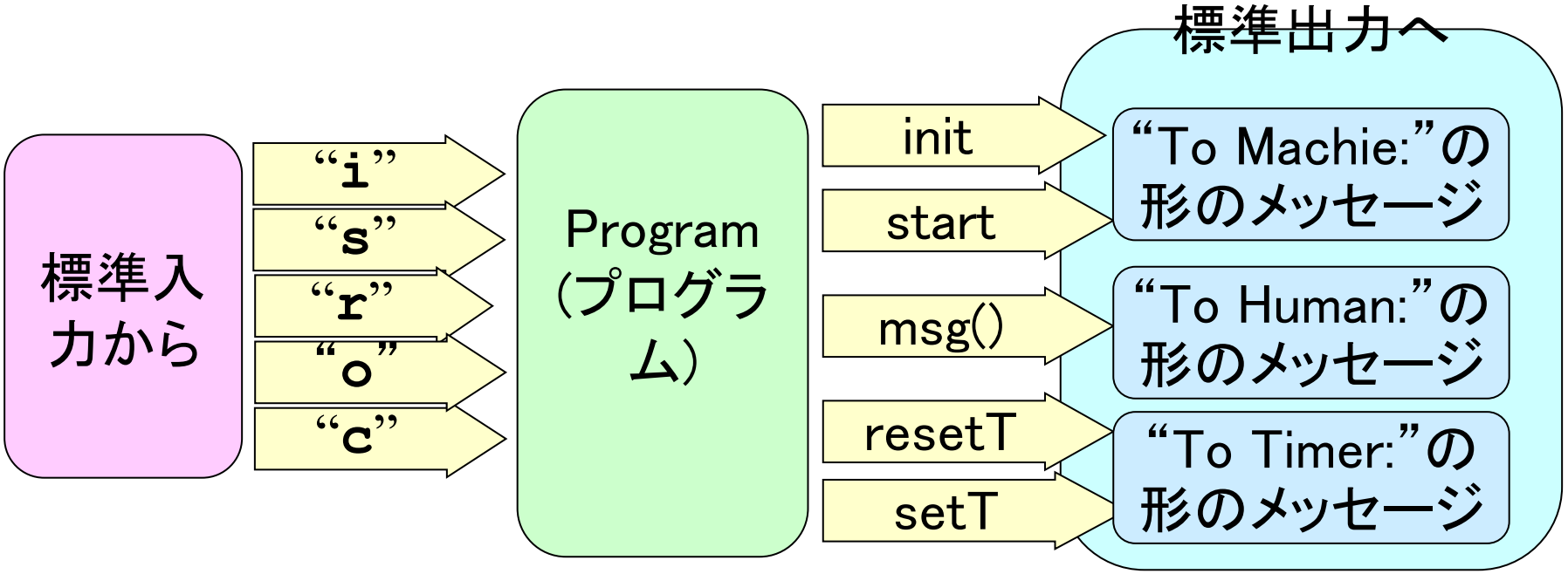




（4．10．1）演習（課題4－3）

■スライド(4.9.1)（もしくは(4.9.2)）に基づき、マシン制御のプログラムを作成してください。

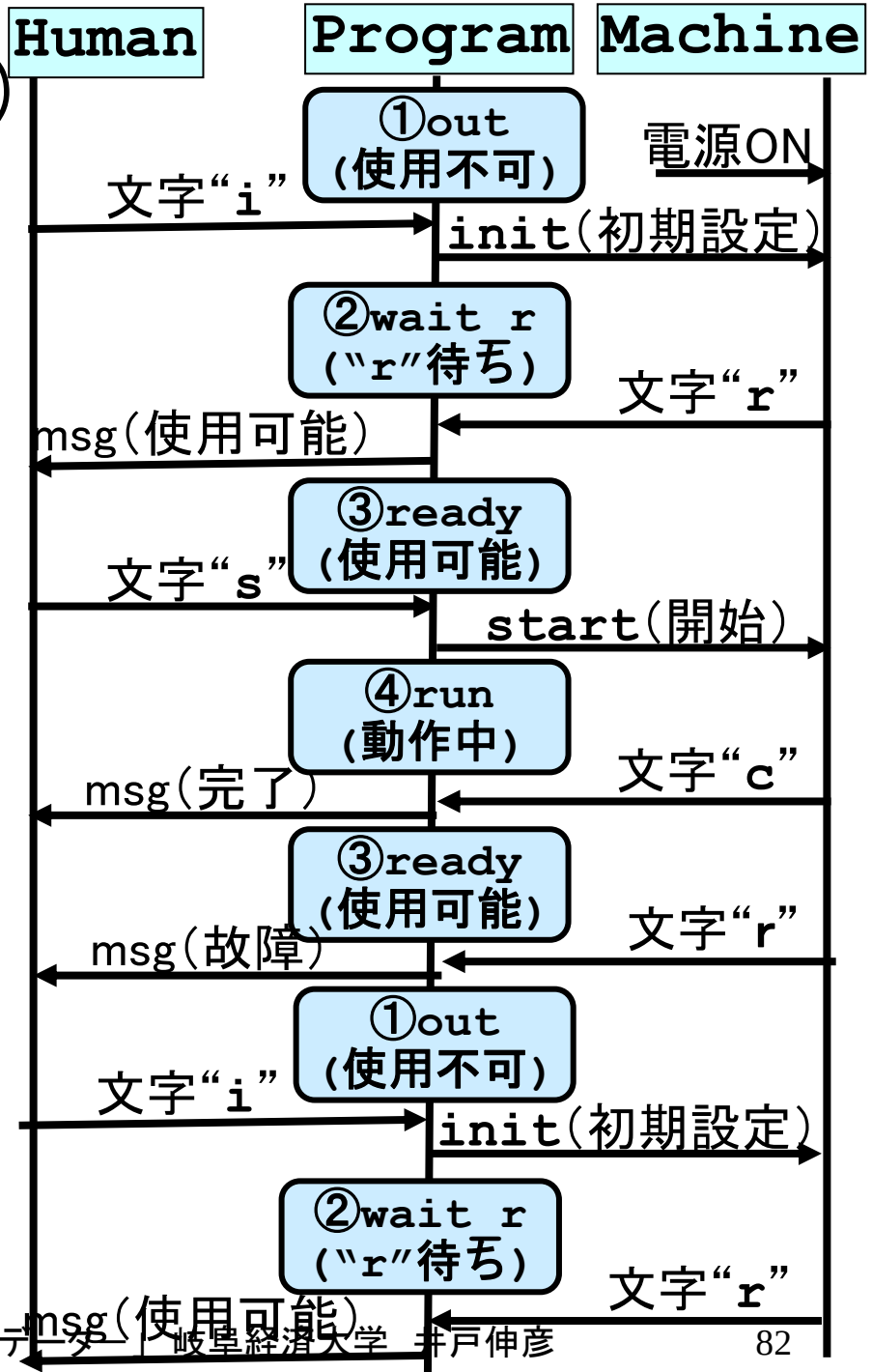
■プログラムへの入力および出力は、標準入出力により擬似することにします。



(4.10.2)ヒント(課題4−3)

■次のような操作イメージとなります。

```
$ java kadai43
i
To Machine:Init
r
To Human:msg(使用可能)
s
To Machine:Start
c
To Human:msg(完了)
r
To Human:msg(故障)
i
To Machine:Init
r
To Human:msg(使用可能)
q
$
```

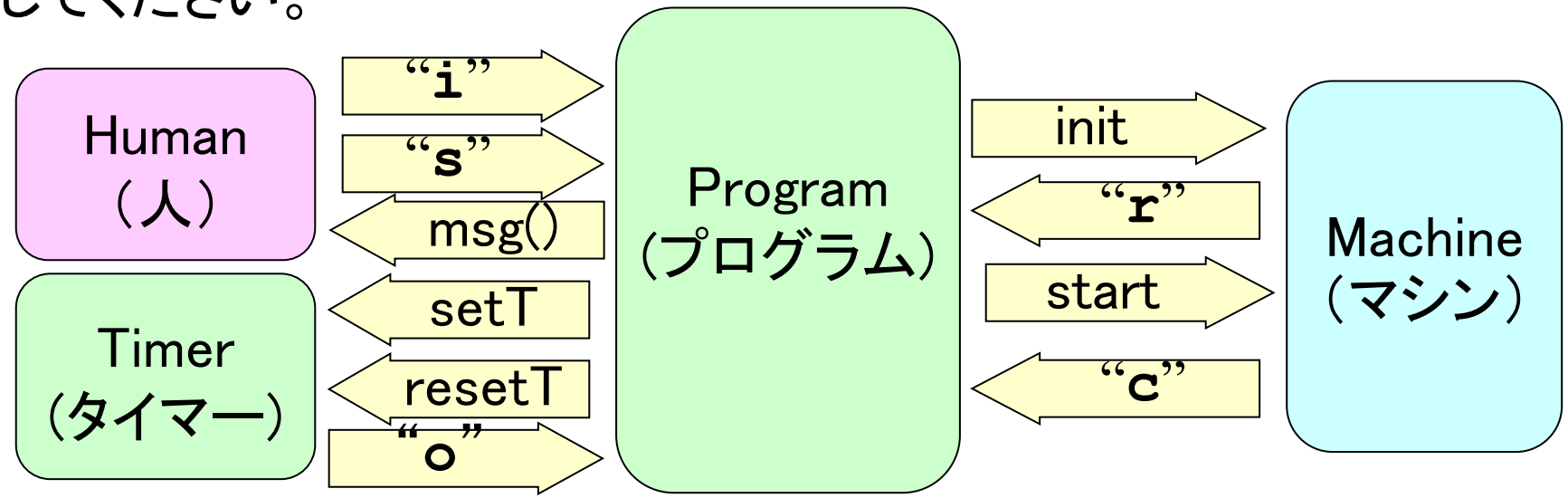


（4．11．1）演習（課題4－4）

■スライド(4.9)の例に、Timer(タイマー)を付け加えます。

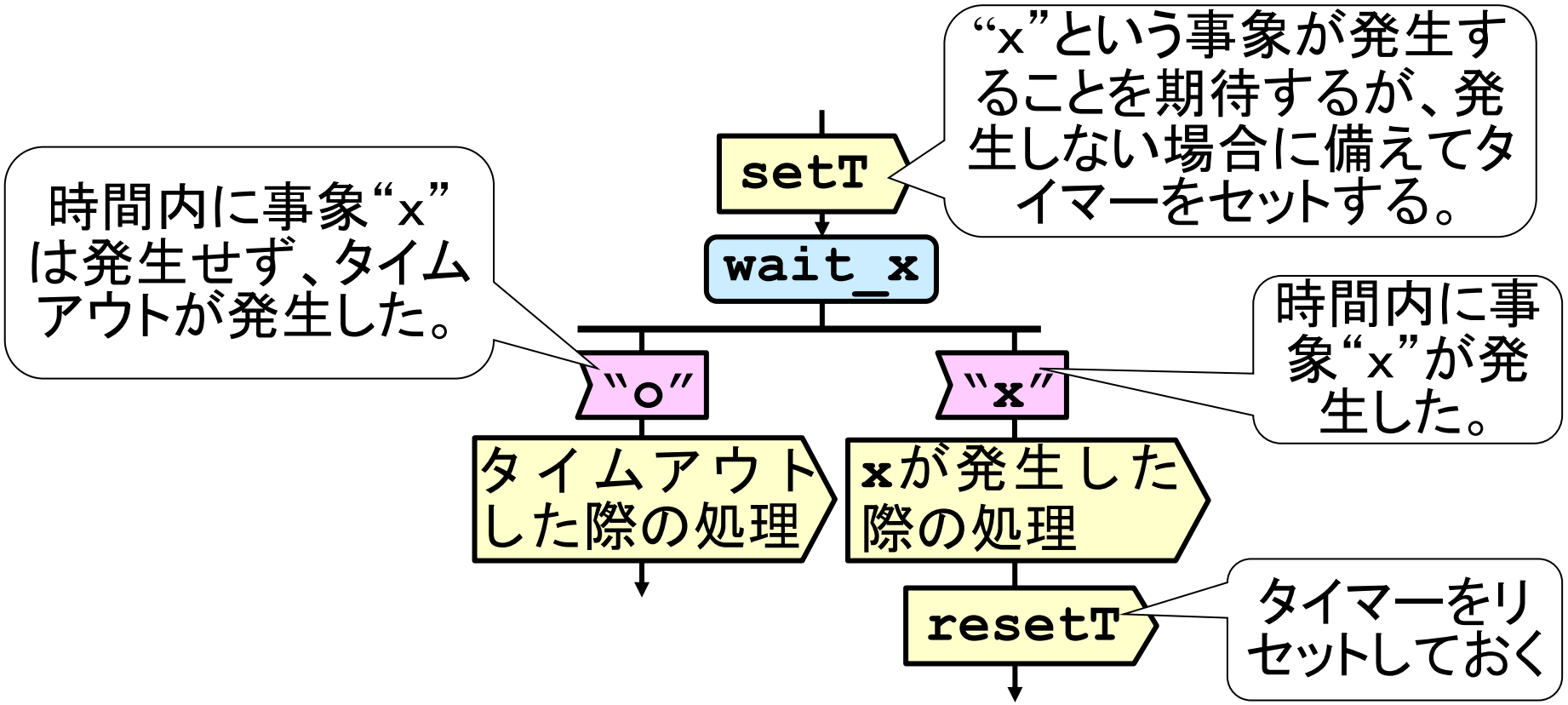
- タイマーにsetを発行すると、一定時間の後タイマーより、文字“o”が入力されます。その前にresetを発行すると、文字“o”の入力は無くなります。
- 上記の機能を用いて、マシンから文字“r”や“c”が一定時間経っても入力されない場合を検出し、この場合をマシンの故障とします。この場合にも、人にはメッセージ(“タイムアウト”)を出します。

■上記の内容を盛り込んで、スライド(4.9.1)の状態遷移図を変更してください。



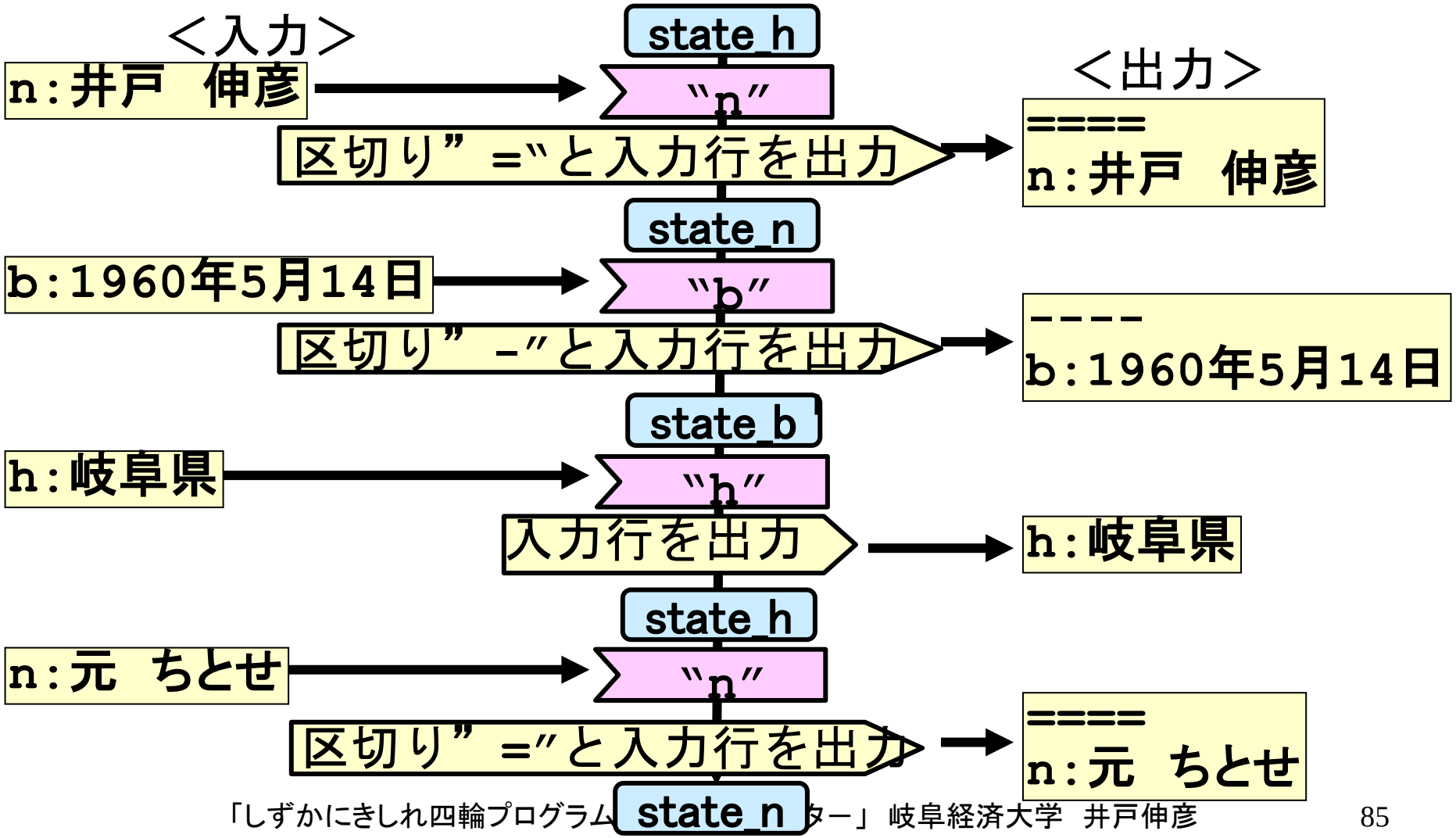
（４．１１．２）ヒント：課題４－４

■タイマーは、下図のように使います。



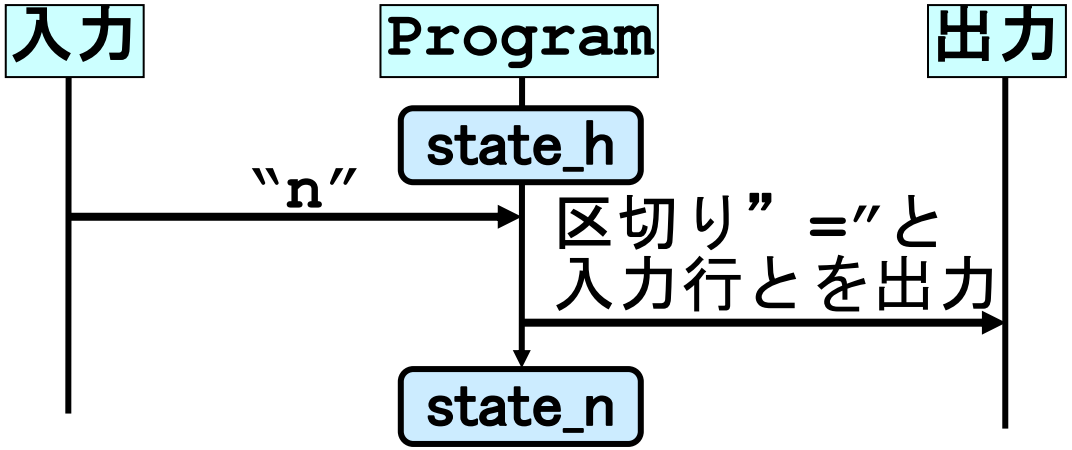
（4．12．1）演習：課題4－5

■フォーマットの問題で考えたスライド(3.3.5)も、シーケンスの変形でした。これをシーケンス図に直してください。



（4．12．2）ヒント：課題４－５

■最初のところを、SDLで書くと、次のようになります。



（４．１３）課題４－６

■次のシーケンス図を状態も含めて書いてください。

- パン工場は、スーパーからパンの注文を受けて、製粉会社に小麦の注文を出した。
- 製粉会社から小麦の入荷があり、パンを焼いて、スーパーにパンの出荷準備が完了したことを連絡した。
- スーパーから出荷するように指示があったので、出荷した。

（４．１４）課題４－７

■課題４－６の状態遷移をシミュレートするプログラムを作成してください。

- パンの注文のイベントは、文字“o”の入力としてください。
- 小麦粉の入荷のイベントは、文字“w”の入力としてください。
- パンの出荷指示のイベントは、文字“s”の入力としてください。
- 状態については、自分で適当な名前を付けてください。次のようにすることも一案です。
  - ◆WAIT_XXX　←　“XXX”を待っている状態

（４．１５）課題４－８

■課題４－６パン工場の状態遷移図について、異常シーケンスを列挙してください。

■列挙した異常シーケンスでの処理案を考えてください。

（４．１６）課題４－９

■次のシーケンス図を状態も含めて書いてください。

- 医師は患者さんから病状説明があったので、レントゲン室にレントゲン撮影依頼をした。
- レントゲン室からレントゲンの撮影結果が来たので、医師は薬局に薬の処方依頼をした。
- 薬局より薬の提出があったので、医師は患者に診断結果を知らせて薬を手渡しした。

（４．１４）課題４－１０

■課題４－９の状態遷移をシミュレートするプログラムを作成してください。

- 患者からの病状説明のイベントは、文字“c”の入力としてください。
- レントゲン室からの撮影結果のイベントは、文字“r”の入力としてください。
- 薬局からの薬の提供のイベントは、文字“m”の入力としてください。
- 状態については、自分で適当な名前を付けてください。次のようにすることも一案です。
  - ◆WAIT_XXX　←　“XXX”を待っている状態

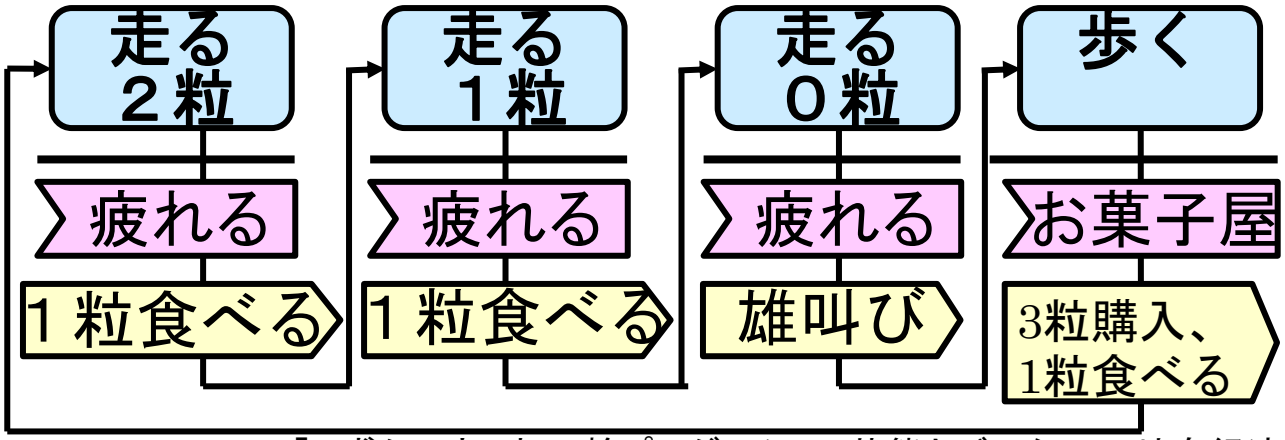
（5．1．1）走る生活

■C君の走る生活を考えます。

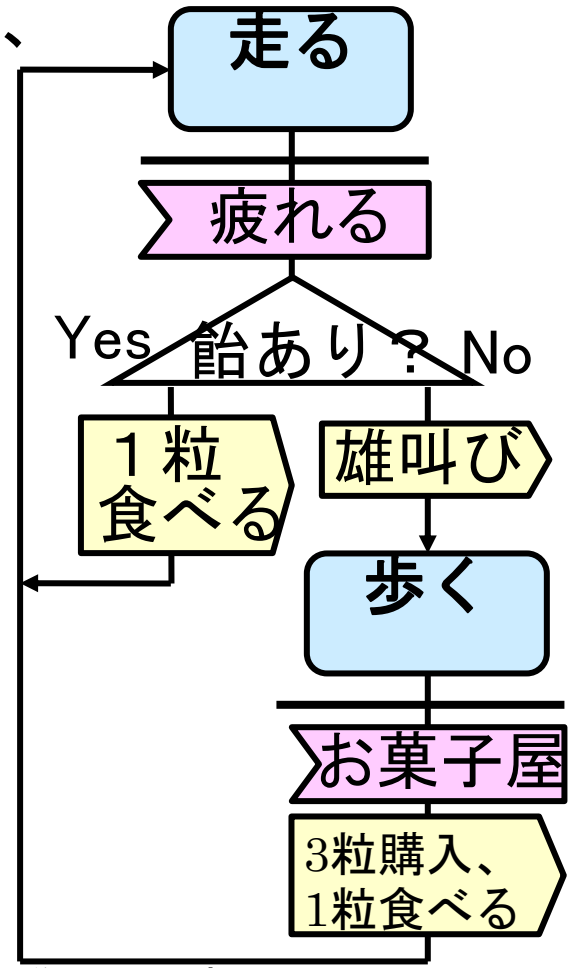
- C君は走っている状態で、疲れてくると、飴（あめ）を1粒食べて、また元気に走り出す。
- 飴を食べようとして、持っていなかった場合、雄叫びをあげてから、歩いてお菓子屋さんまで行く。
- 歩いている状態で、お菓子屋さんに着くと、3粒入りの飴を買い、1粒食べて、また走り出す。

■AとBの2つの状態遷移を考えます。

＜A：4状態＞



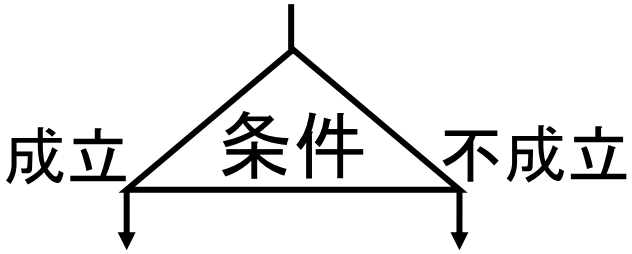
＜B：2状態＞



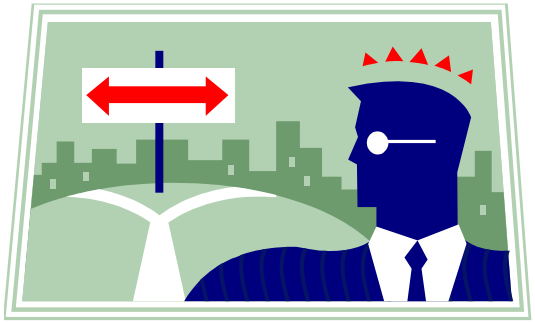
（５．１．２）分岐要素

■前スライドの状態遷移B（2状態）では、分岐要素が使われています。

■分岐要素では、三角形内に書かれた条件の成否により、アクションと次状態とが分かれることになります。



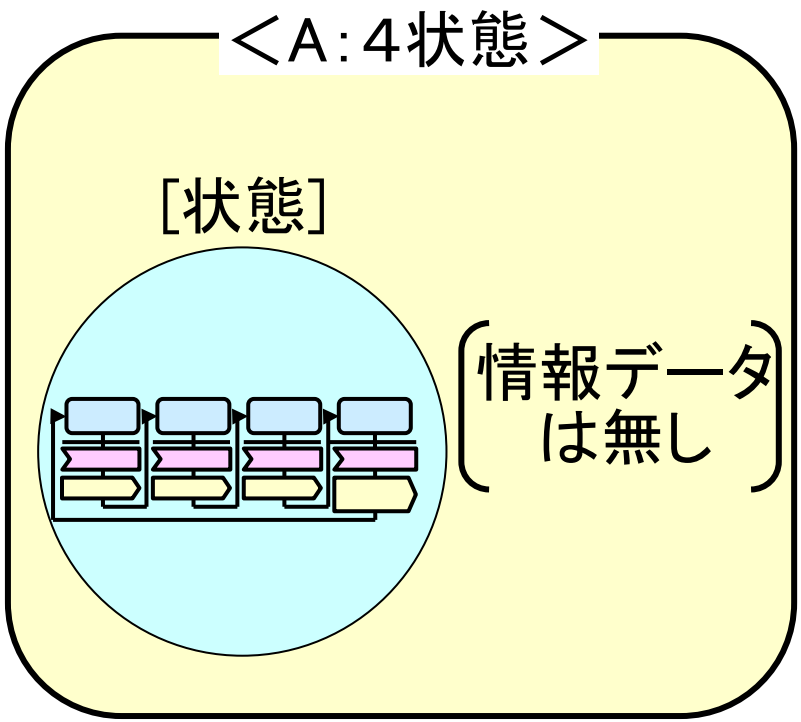
■このような分岐要素は、状態遷移図で一般的に使われます。どのような場合に用いられるのでしょうか？

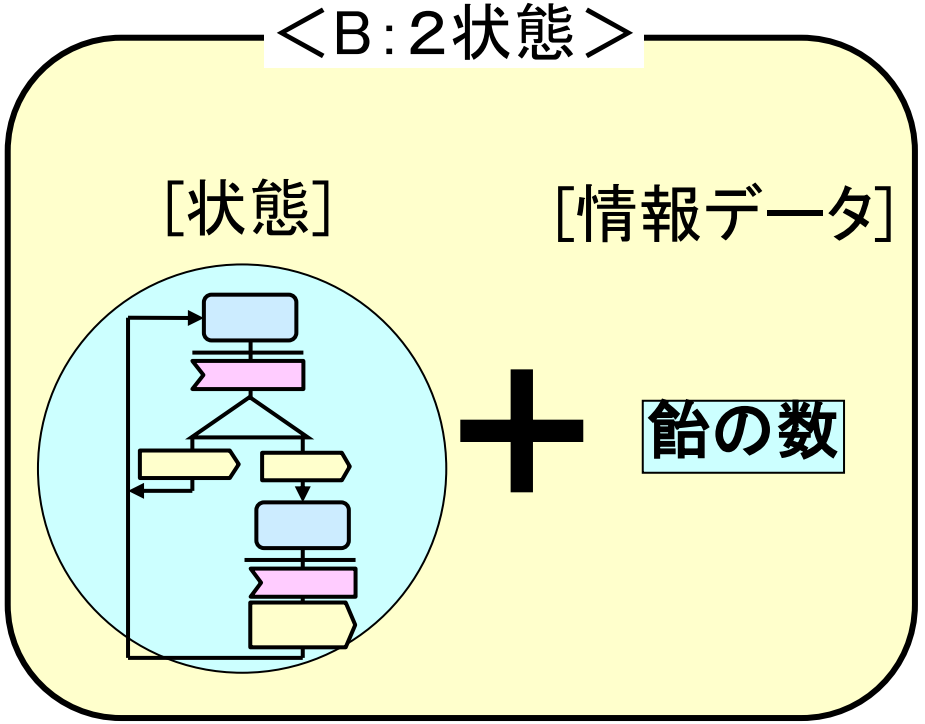


（５．１．３）飴1000粒　ー情報データー

■飴が1000粒入りであった場合、A（４状態）は1001状態となり、状態遷移と捉えることは現実的ではありません。

■飴の数は、状態ではなく、“情報データ”（スライド(1.3)）であると捉えた方が良い訳です。

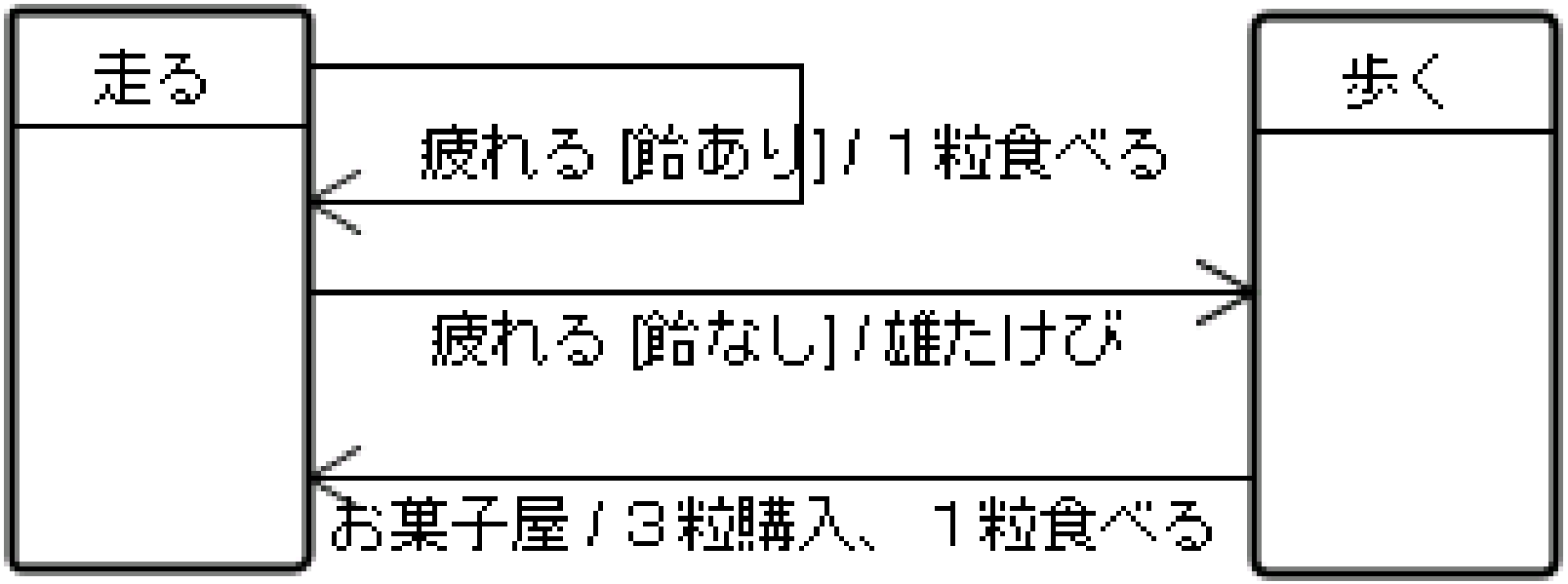




（５．２．１）Judeでの分岐要素

■Judeでは、次のように書きます。

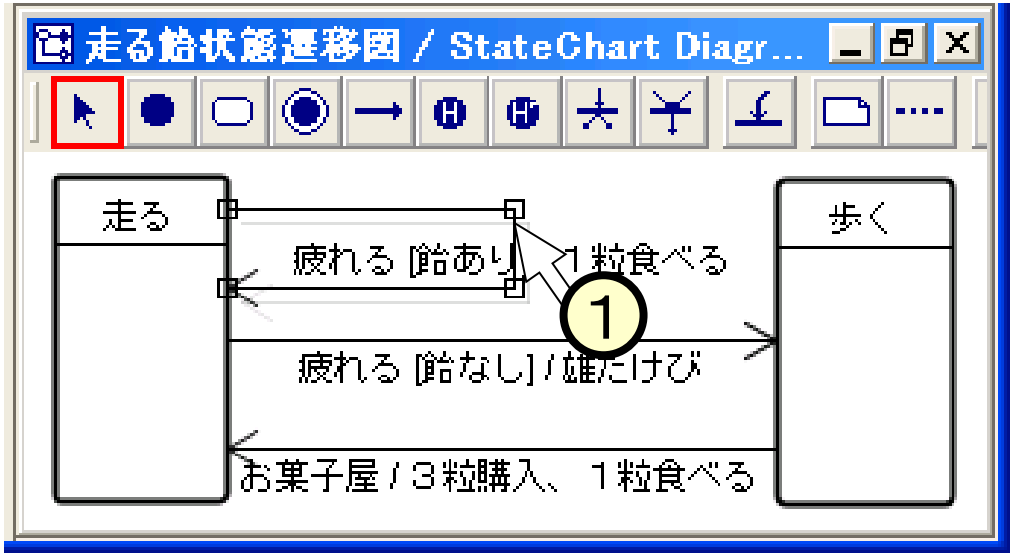
- “[飴あり]”、“[飴なし]”と書かれている部分を、ガード条件と呼びます。
- 上側の遷移は、“疲れる”というイベントが起こったとき、“[飴あり]”というガード条件が満たされている場合、１粒食べるというイベントの後に、走るという状態へ移ることを示します。



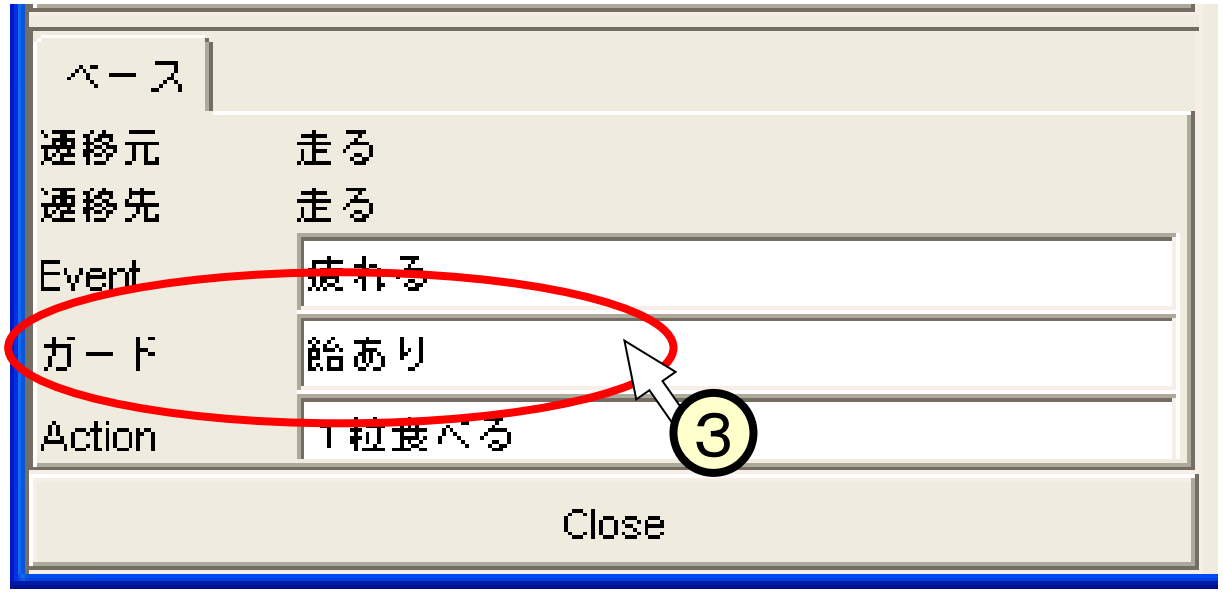
（5．2．2）ガード条件の作成

■イベントやアクションと同様に作成します。

■遷移をクリック（①）して選択します。



■プロパティ・ビュー中で、[ガード]の欄に入力（②）します。



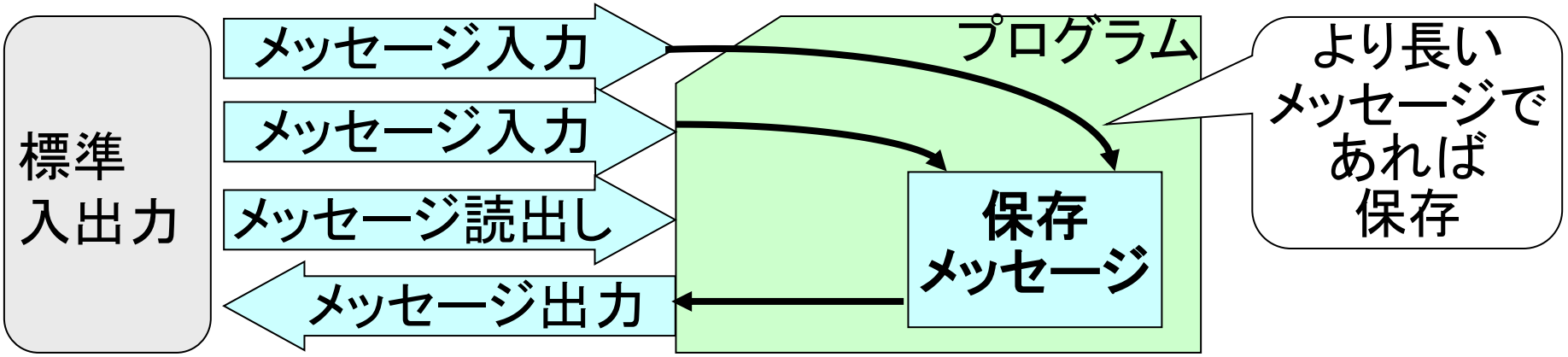
（5．2．3）演習（課題5－1）

■スライド(5.2.1)の状態遷移図を作成してください。

（５．３．１）メッセージ保存　－その１－

■次のようなプログラムを考えます。

- メッセージ入力を受け付けて、入力・保存・読出しを行う機能を持ちます。
- 受け付けたメッセージが、現在保存されているメッセージよりも長い場合（あるいはメッセージが保存されていない場合）は、保存するメッセージを入れ替えます。同じか短い場合には、受け付けたメッセージを廃棄します。
- メッセージ読出しを受け付けた場合、保存しているメッセージを出力し、該メッセージを廃棄します。保存しているメッセージが無い場合は、“no message”と出力します。



（５．３．２）メッセージ保存　－その２－

■次のようなプログラムを考えます（つづき）。

- 前スライド(5.3.1)に記した機能は、ONの入力を受け付けて、動き出します。OFFの入力を受けると、ON以外のすべての入力を無視する状態になります。ON/OFF切り替え時には、その旨を出力します。
- 停止の際、保存していたメッセージは廃棄されます。
- プログラムへの入力は、次の形式をしています（メッセージの保存の際は、”m:”を含めてOKです）。

#	入力	形式（文字列）
1	ON	n
2	OFF	f
3	メッセージ入力	m:メッセージ
4	メッセージ読出し	r

（５．３．３）動かして見る

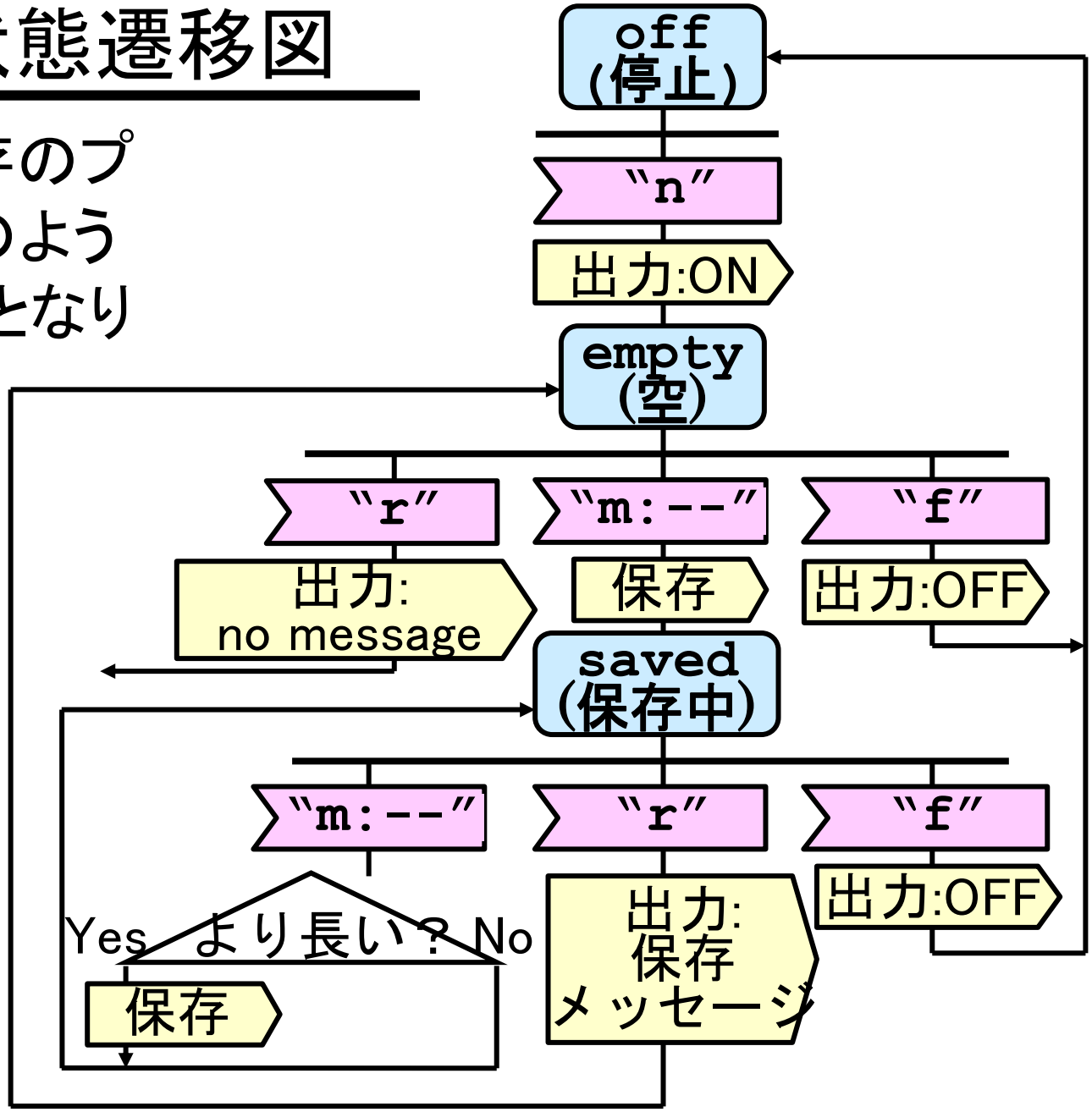
■メッセージ保存のプログラムを、井戸のネットワークドライブからダウンロードして、動かしてみます。

- 井戸のネットワークドライブから、“マイドキュメント”の下の“javawork”のフォルダに、次のファイルをコピーしてください。
  - ◆SaveMessage.java
- eclipseにインポートして動かします。
- 次のようなイメージで動作します。

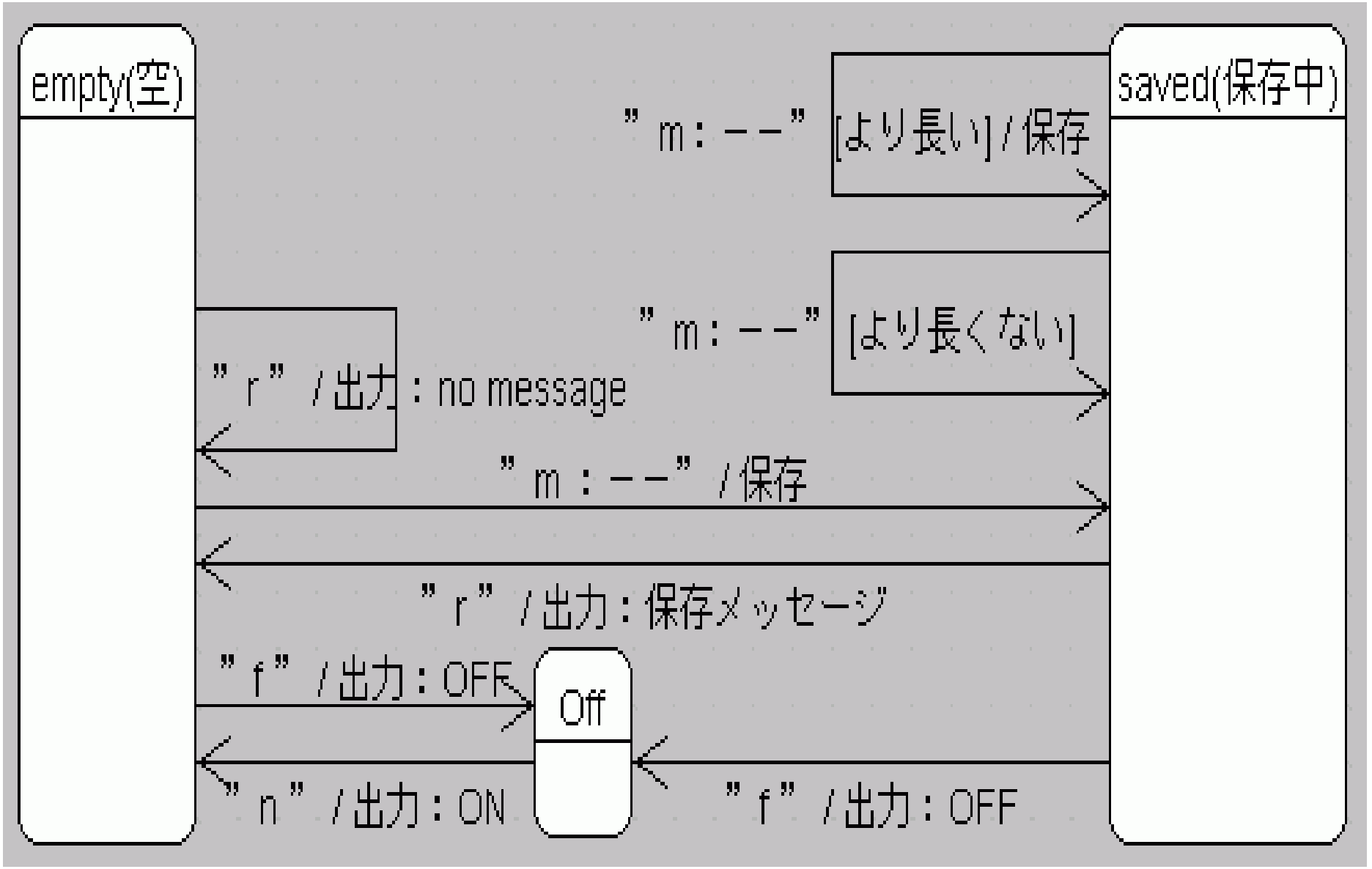
```
> cd javawork              ← javaworkのフォルダに移動
> java saveMesage          ← saveMessage.classのプログラムを起動
n                          ← ここから、”n”,”f”,”r”,”m:--”を入力
ON
m:hello
r
m:hello
q                          ← “q”を入力すると終了
>
```

（５．３．４）状態遷移図

■メッセージ保存のプログラムはどのような状態遷移図となります。



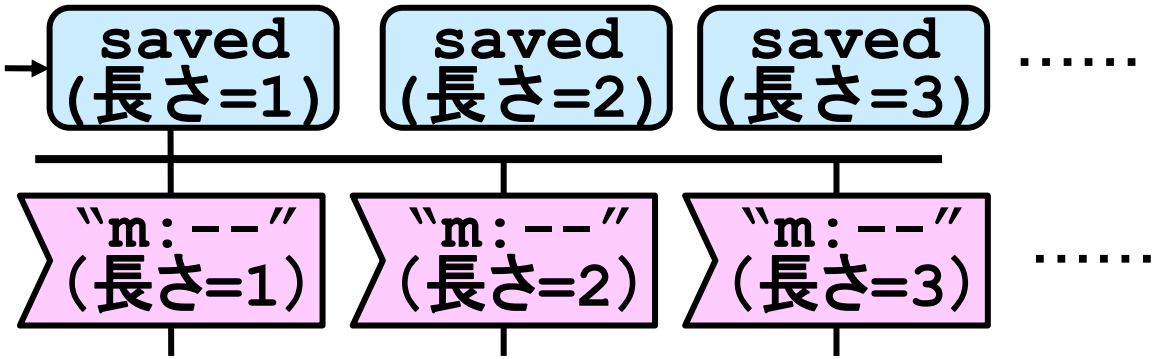
（５．３．５）状態遷移図　－Jude版－



（５．３．６）やってみないと解らない

■メッセージ保存の例では、メッセージの長さを比較しないことには、後の処理が決められません。

■スライド(5.1.1)＜A：４状態＞のような状態遷移を考えようとしても、状態ばかりでなく、イベントもメッセージの長さごとに定義しなければならなくなります。



■すなわち、“比較して見なければわからない”ことですから、分岐が必要となってきます。

（5．3．7）プログラム（1／3）

```
import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;
public class SaveMessage{
  static final int OFF = 0;
  static final int EMPTY = 1;
  static final int SAVED = 2;
  static int state=OFF;
  static String savedMessage = "";
  public static void main(String args[]) throws
IOException{
    String event;
    BufferedReader cin = new BufferedReader(new
InputStreamReader(System.in));
    while((event=cin.readLine())!=null){
      if(event.startsWith("q")){ break;}
```

“OFF”と書けば、“0”を意味するようする。他も同様

これ(=state)が“状態”。最初は“OFF”。

保存しておくメッセージ。

コンソールから行を読み取れるようにする。

読み取りを繰り返し、その行をイベントとする

イベントが“q”で始まるならば、終了する。

（5．3．8)プログラム(2／3）

```
        if(state==OFF){
          if(event.startsWith("n")){
            System.out.println("ON¥n");
            state=EMPTY;}
        }else if(state==EMPTY){
          if(event.startsWith("r")){
            System.out.println("no
 message¥n");
            state=EMPTY;
          }else if(event.startsWith("m")){
            savedMessage=event;
            state=SAVED;
          }else if(event.startsWith("f")){
            System.out.println("OFF¥n");
            state=OFF;
          }
```

状態が“OFF”の場合

状態が“EMPTY”の場合

（５．３．９)プログラム(３／３）

```
        }else if(state==SAVED){
          if(event.startsWith("m")){
            if(event.length()>savedMessage.length()){
              savedMessage=event;}
            state=SAVED;
          }else if(event.startsWith("r")){
            System.out.println(savedMessage+"¥n");
            state=EMPTY;
          }else if(event.startsWith("f")){
            System.out.println("OFF¥n");
            state=OFF;
          }
        }
      }
    }
}
```

状態が“SAVED”の場合

メッセージの長さを比較する。

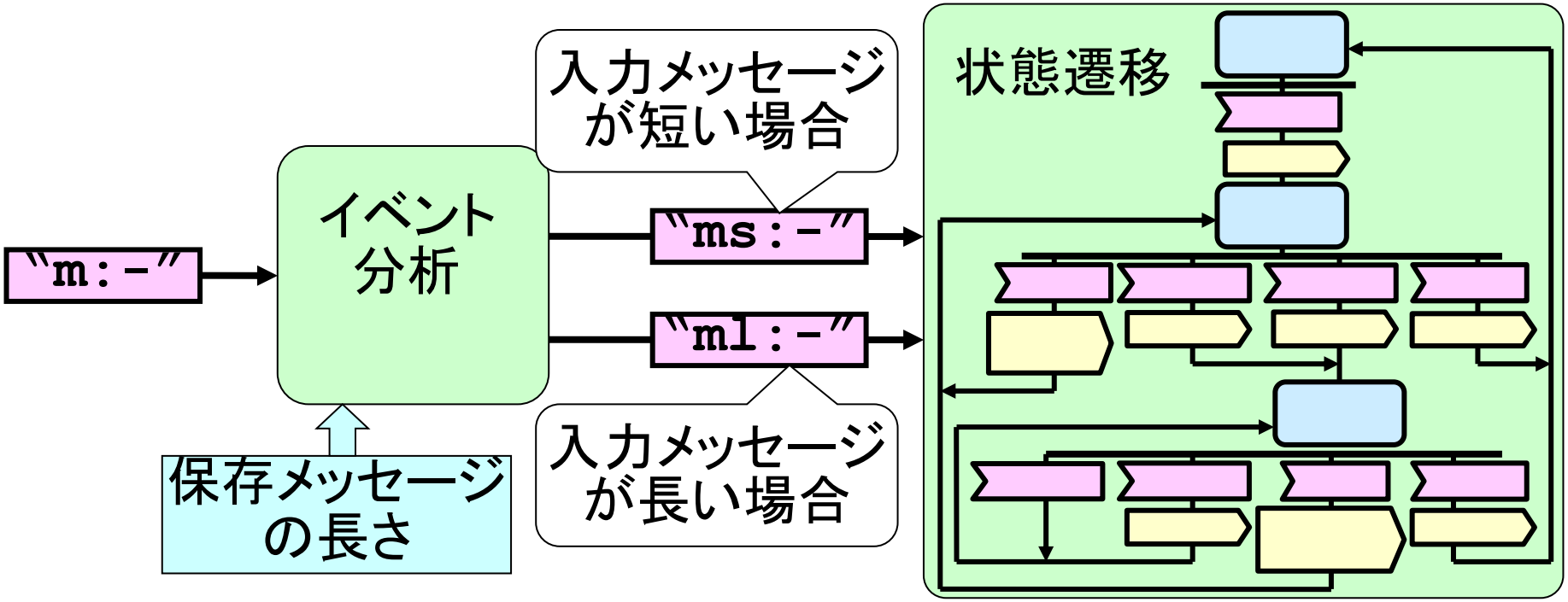
(5．3．10)プログラム-JavaScript-

```
<html>
<head><title>s_5_3_10</title></head>
<body onload='setInterval(main,100);'>
<script type="text/javascript">
const OFF= 0;
const EMPTY=1;
const SAVED= 2;
var state=OFF;
var savedMessage='';
function main(){
 var in_c=document.getElementById('text0');
 var msg_c=document.getElementById('text1');
 var out_c=document.getElementById('p0');
 var event=in_c.value;
 if(event=='')return;
 if(state==OFF){
   if(event=='n' ){
     out_c.innerText='ON'; state=EMPTY;
   }
 }else if(state==EMPTY){
   if(event=='r' ){
     out_c.innerText='no message.'; state=EMPTY;
   }else if(event=='m'){
     savedMessage=msg_c.value;
     msg_c.value='';
     state=SAVED;
   }else if(event=='f'){
     out_c.innerText='OFF';
     state=OFF;
   }
 }else if(state==SAVED){
   if(event=='m'){
     if(msg_c.value.length>savedMessage.length){
       savedMessage=msg_c.value;
     }
     state=SAVED;
   }else if(event=='r' ){
     out_c.innerText=savedMessage;
     state=EMPTY;
   }else if(event=='f' ){
     out_c.innerText='OFF'; state=OFF;
   }
 }
 in_c.value='';
}
</script>
<form>
command:<input type='text' id='text0' /><br />
message:<input type='text' id='text1' />
</form>
<p id='p0'></p>
</body>
</html>
```

（５．４．１）イベント分析

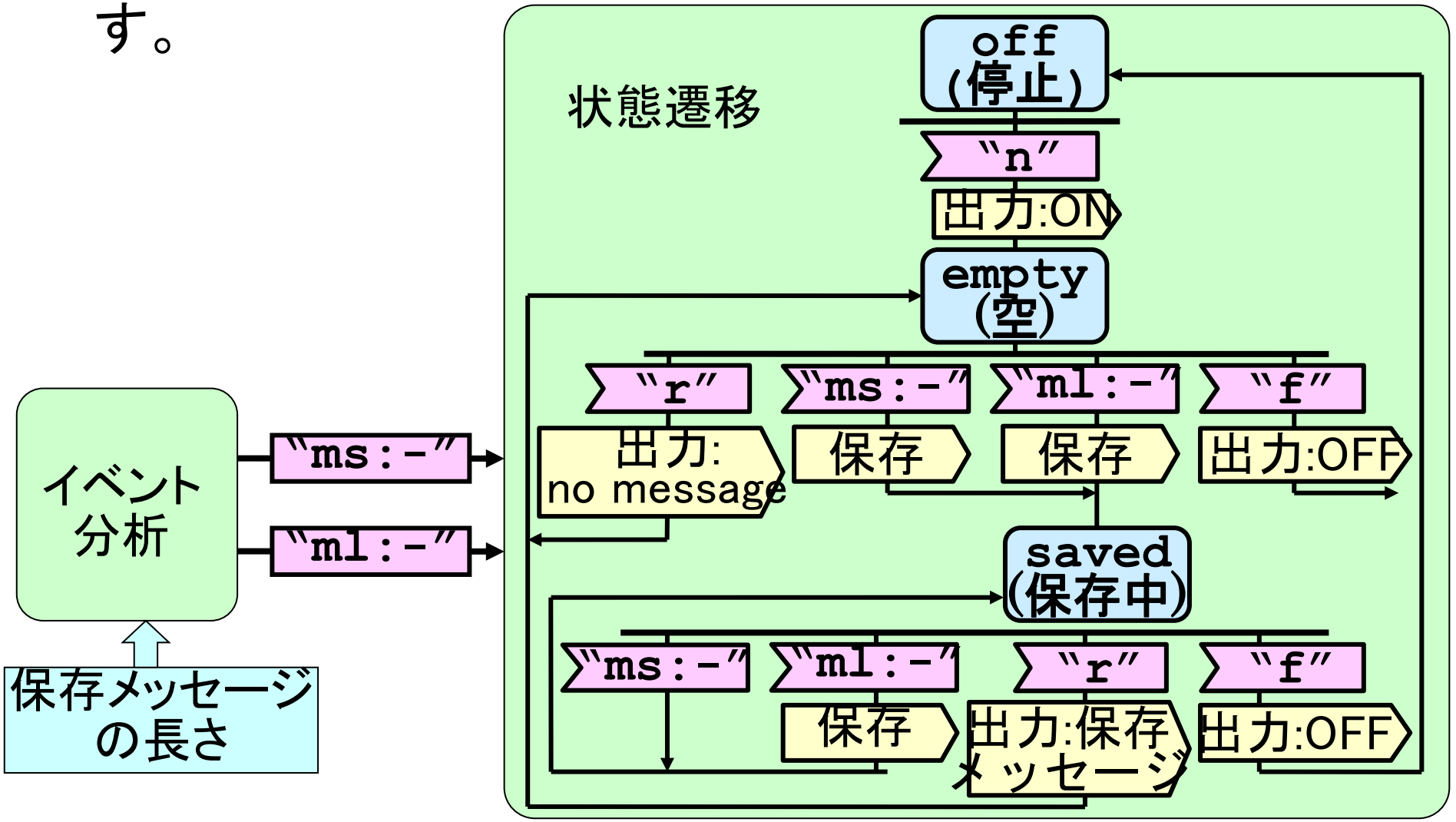
■スライド(5.3.4)以外に方法は無いかというと、そうでもありません。

■状態遷移の前に、入力メッセージと保存メッセージとの長さを比較して、その結果によりイベントを生成するプログラム（イベント分析）を設ける方法もあります。



（５．４．２）イベント分析ありの状態遷移

■イベント分析を設けると、図のような状態遷移となります。

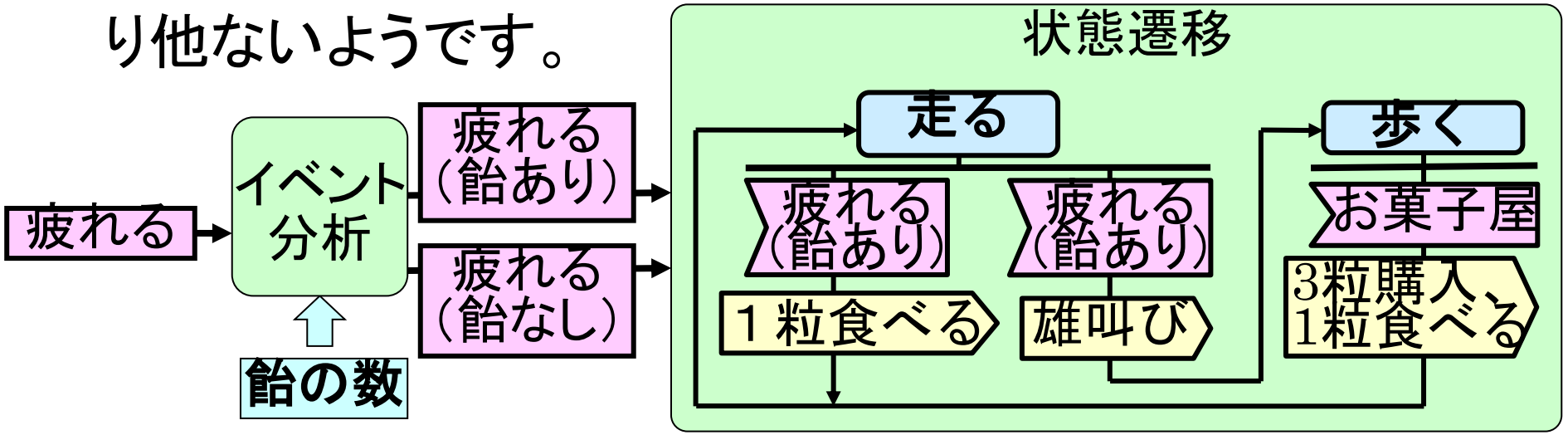


（5．4．3）分岐は必要か？

■イベント分析を用いれば、スライド(5.1.1)の<B:2状態>の状態遷移図も、分岐を無くすことは出来ます。

■しかしながら、イベント分析を設けると、イベントの数が増えてしまう問題があります。このことから、分岐を徹底してなくすと、かえって煩雑になる場合もあります。

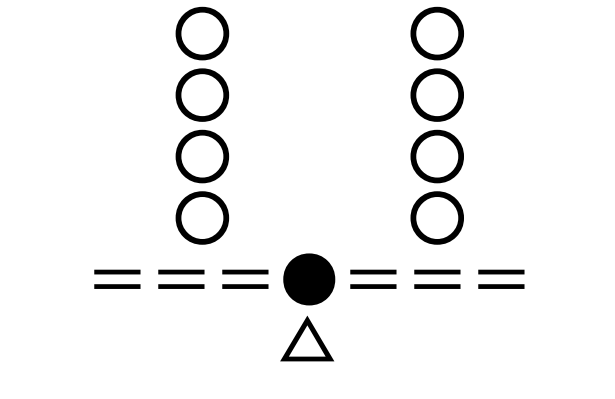
■結局のところ、他の設計の問題と同様に、場合により“分岐”とするか“イベント分析”とするかを考えていくより他ないようです。

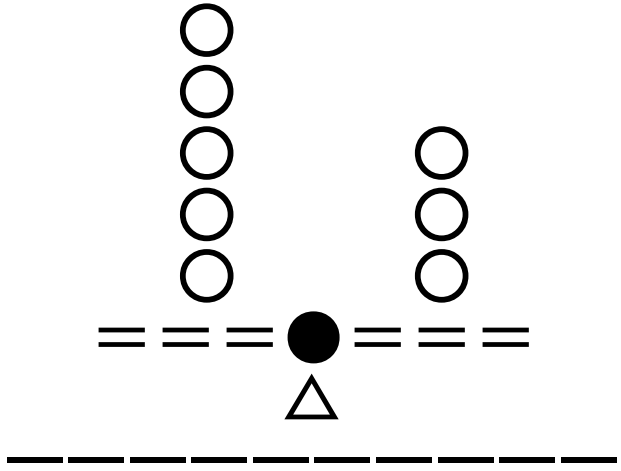


（５．５．１）演習（課題５－２）

■次の問題を考えます。

- はかりの両側に4つずつの玉が乗っています。
- “t”と入力すると、玉はひとつずつ左へ移っていきます。
- “c”と入力すると、玉の移る方向が変わります。
- 片方に6個より多くの玉が乗ると、はかりは壊れ、“broken”と出力して、終了となります。





■このプログラムの状態遷移図を作成してください。

（５．５．２）演習：動かして見る

■メッセージ保存のプログラムを、井戸のネットワークドライブからダウンロードして、動かしてみます。

- 井戸のネットワークドライブから、“マイドキュメント”の下の“javawork”のフォルダに、次のファイルをコピーしてください。

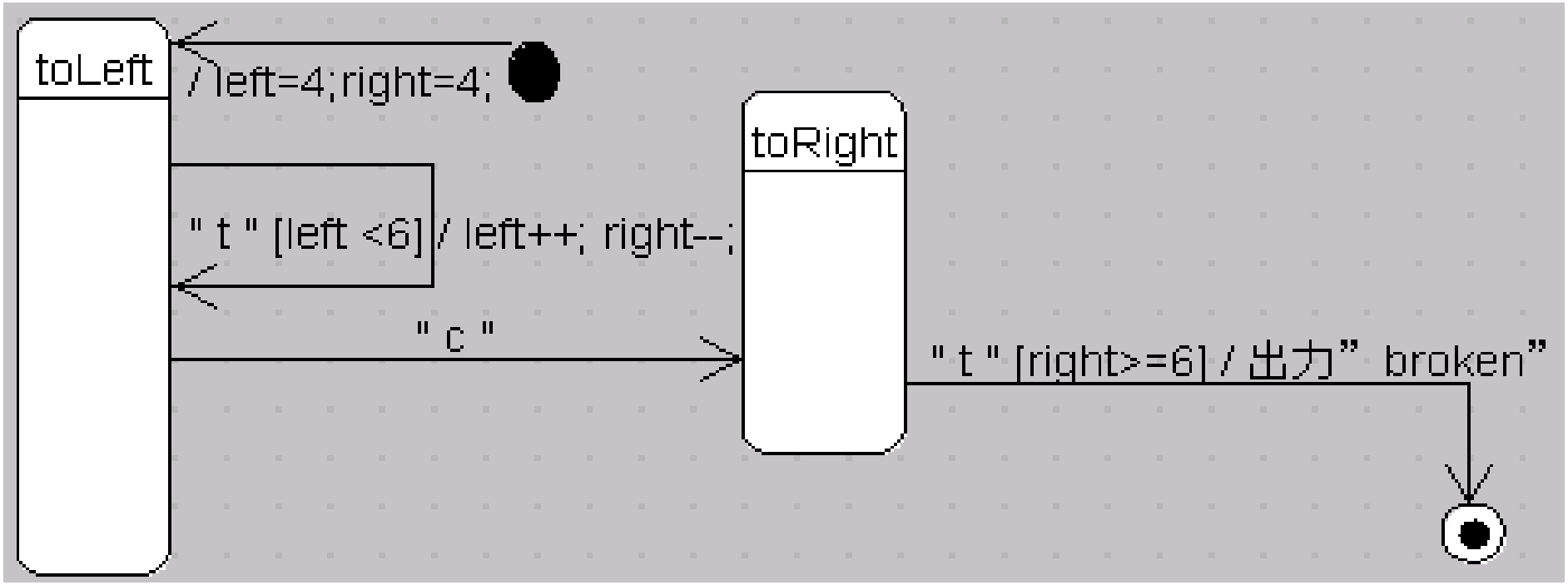
  Balance.java

- [start]–[すべてのプログラム]–[アクセサリ]–[コマンドプロンプト]を選択してください。
- 次のようなイメージで動作します。

```
> cd javawork          ← javaworkのフォルダに移動
> java balance         ← balance.classのプログラムを起動
t                      ← ここから、”t”,”c”,”q”を入力
q                      ← “q”を入力すると終了
>
```

（５．５．３）ヒント：課題５－２

- 左の玉の数を“left”、右の玉の数を“right”とします。
- 下の未完の状態遷移図に遷移要素を付け加えて完成させてください。
  - ●は開始を、◉は終了を、それぞれ表す記号です。
  - left++；は、leftに1を加えることを表します。



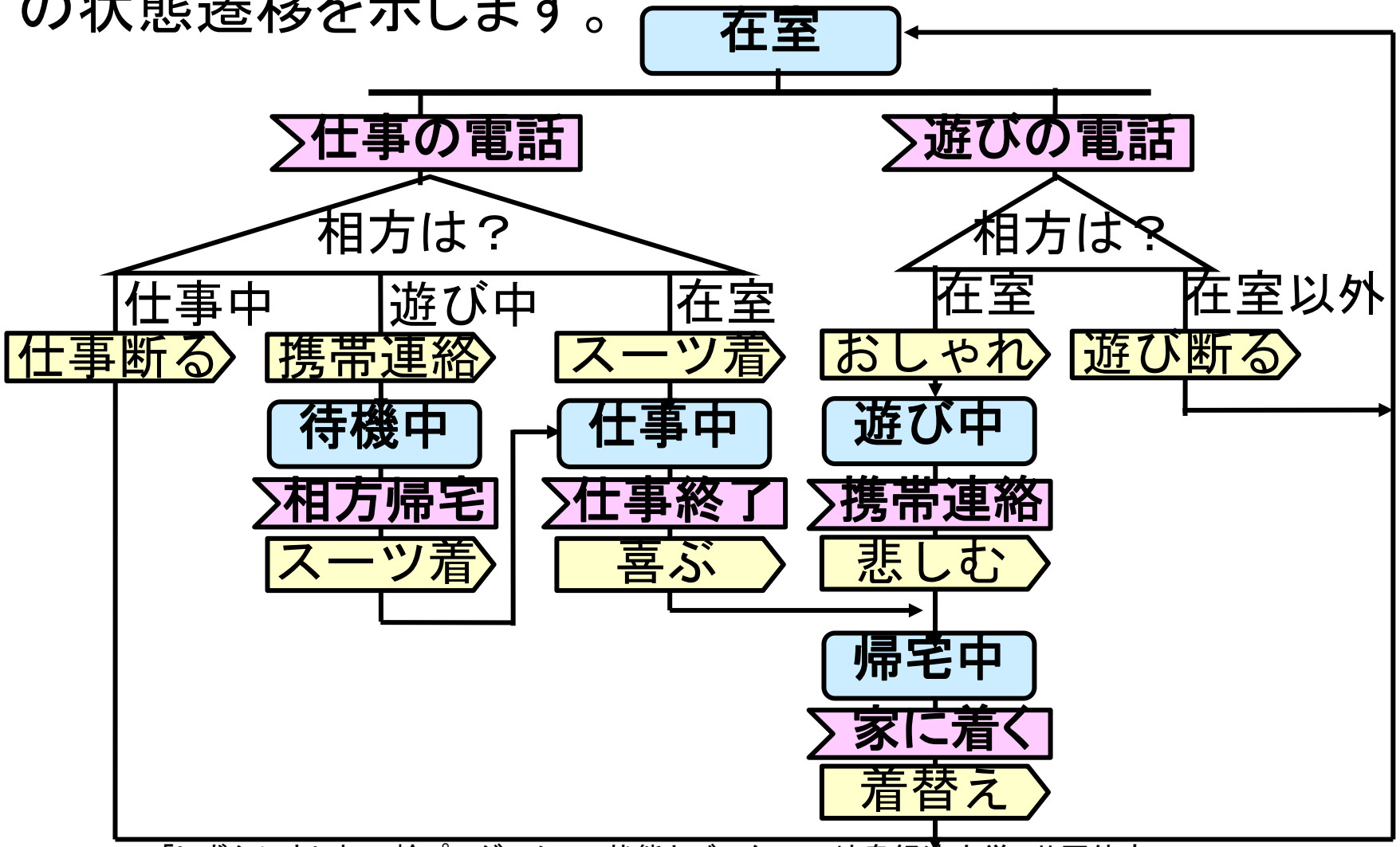
（６．１）ルームメイト　－その１：密接－

■AさんとBさんは、ルームメイトです。ちょっと風変わりですが、次のルールで生活しています。

- 二人とも部屋を空けることは無いようにしている。
- 二人とも部屋にいるとき
  - 仕事の電話がかかって来ると、一人はスーツに着替えて出かける。
  - 遊びの電話がかかって来ると、一人はおしゃれをして出かける。
- 一人で部屋にいるとき
  - 相方が仕事中のとき
    - 仕事の電話がかかってくると、仕事を断る。
    - 遊びの電話がかかってくると、遊びを断る。
  - 相方が遊び中のとき
    - 仕事の電話がかかってくると、相方に携帯電話で連絡をとり、帰宅するように伝える。相手が返ってきたら、スーツに着替えて出かける。
    - 遊びの電話がかかってくると、遊びを断る。
- 仕事で出かけた人は、仕事が終わると、喜んで帰宅の途に付く。
- 遊びに出かけた人は、
  - 携帯連絡を受けたら、悲しんで帰宅の途につく。
- 帰宅した人は着替えをする。

（6．1．1）一人の状態遷移図

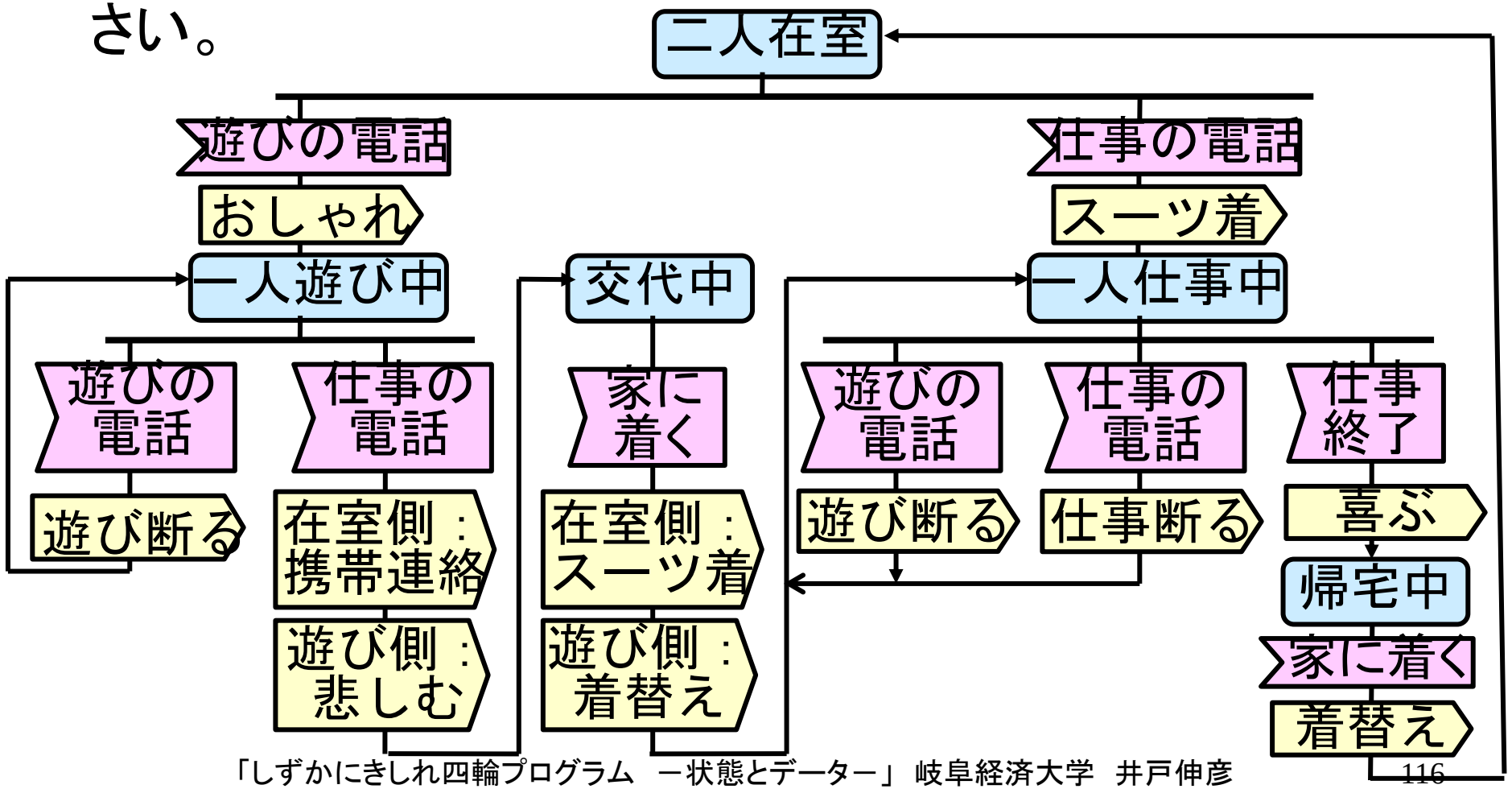
■スライド(6.1)に示したルームメイトの、“一人”についての状態遷移を示します。



（6．1．2）二人の状態遷移図

■スライド(6.1)に示したルームメイトの、“二人”についての状態遷移を示します。

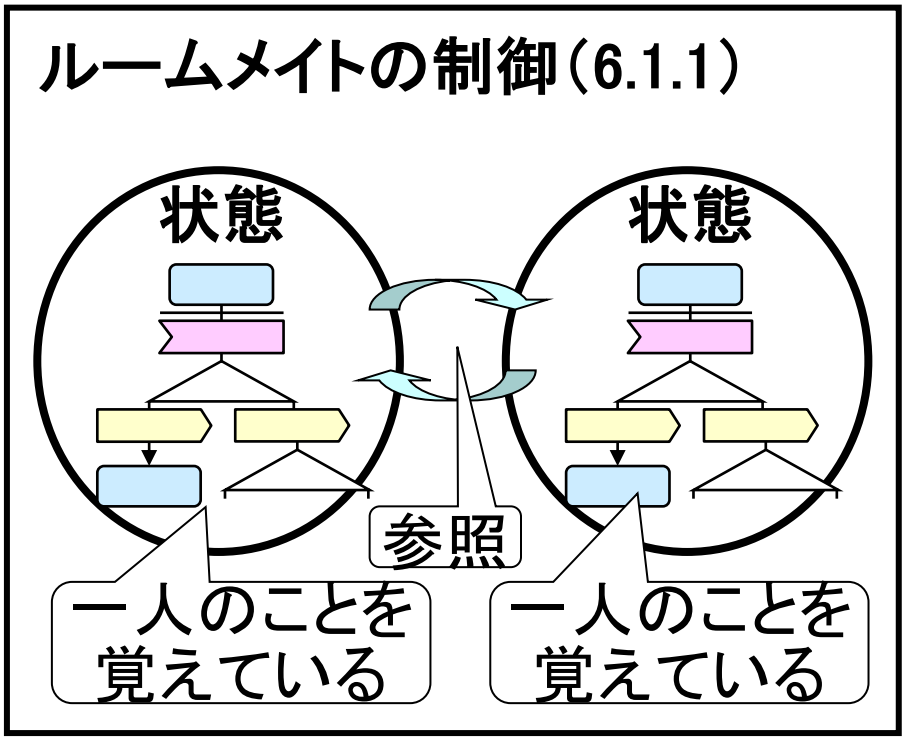
■スライド(6.1.1)と同じことを示しています。確認してください。

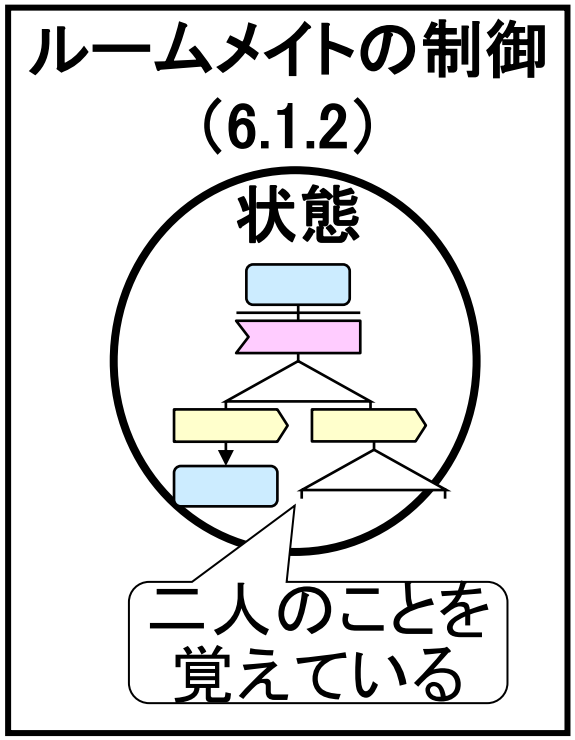


（6．1．3）2つの状態遷移図

■(6.1.1)では、ルームメイトの制御について、2つの状態データを持っています。

■これに対して、(6.1.2)では、ひとつの状態データにより制御を行います。





（６．１．４）どちらが良いか？

■スライド(6.1.2)「二人の状態遷移図」のほうが、スライド(6.1.1)「一人の状態遷移図」解りよいと感じられたのではないでしょうか？

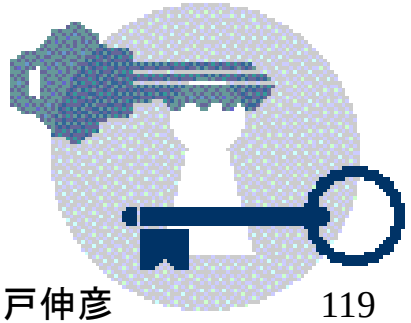
- (6.1.1)では、二人とも仕事中であることがあるのか無いのか、直ぐには解らない。(6.1.2)では、全体の動きがもれなく理解できる。
- (6.1.1)では、分岐が入っており煩雑であるが、(6.1.2)では分岐が無い。

■それでは、(6.1.2)式のほうがいつでも良いかというと、そういう訳ではありません。次の例を見てみましょう。

（６．２）ルームメイト　－その２：疎遠－

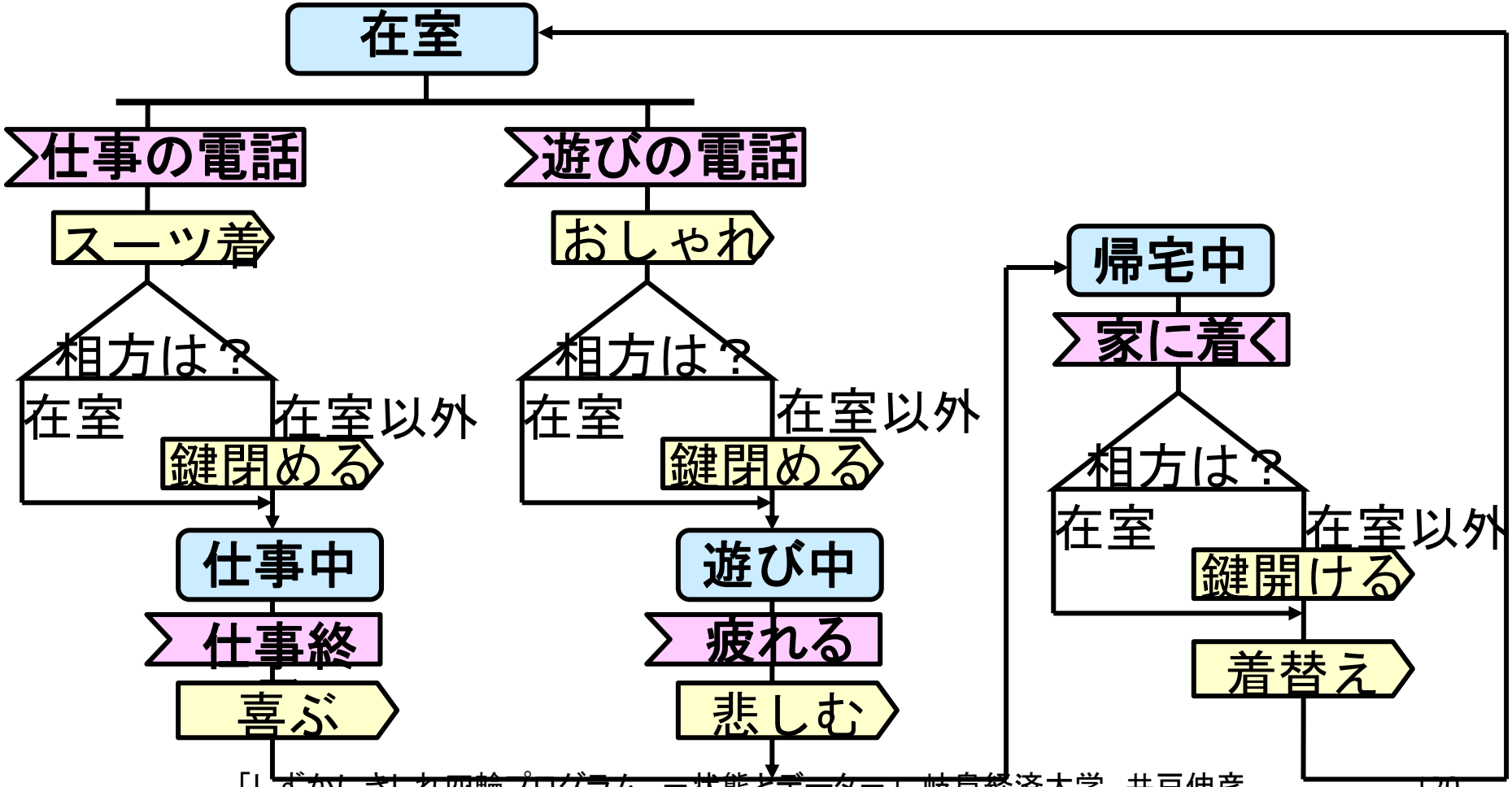
■CさんとDさんは、ルームメイトです。

- 二人とも、
  - ◆仕事の電話がかかって来ると、スーツに着替えて出かける。
  - ◆遊びの電話がかかって来ると、おしゃれをして出かける。
- 出かけるときは、相方がいなければ、鍵を閉める。
- 仕事で出かけた人は、仕事が終わると、喜んで帰宅の途に付く。
- 遊びに出かけた人は、疲れると悲しんで帰宅の途に付く。
- 帰宅した人は、相方が在宅していないなら、鍵を開けてる。
- 帰宅した人は、着替えをする。



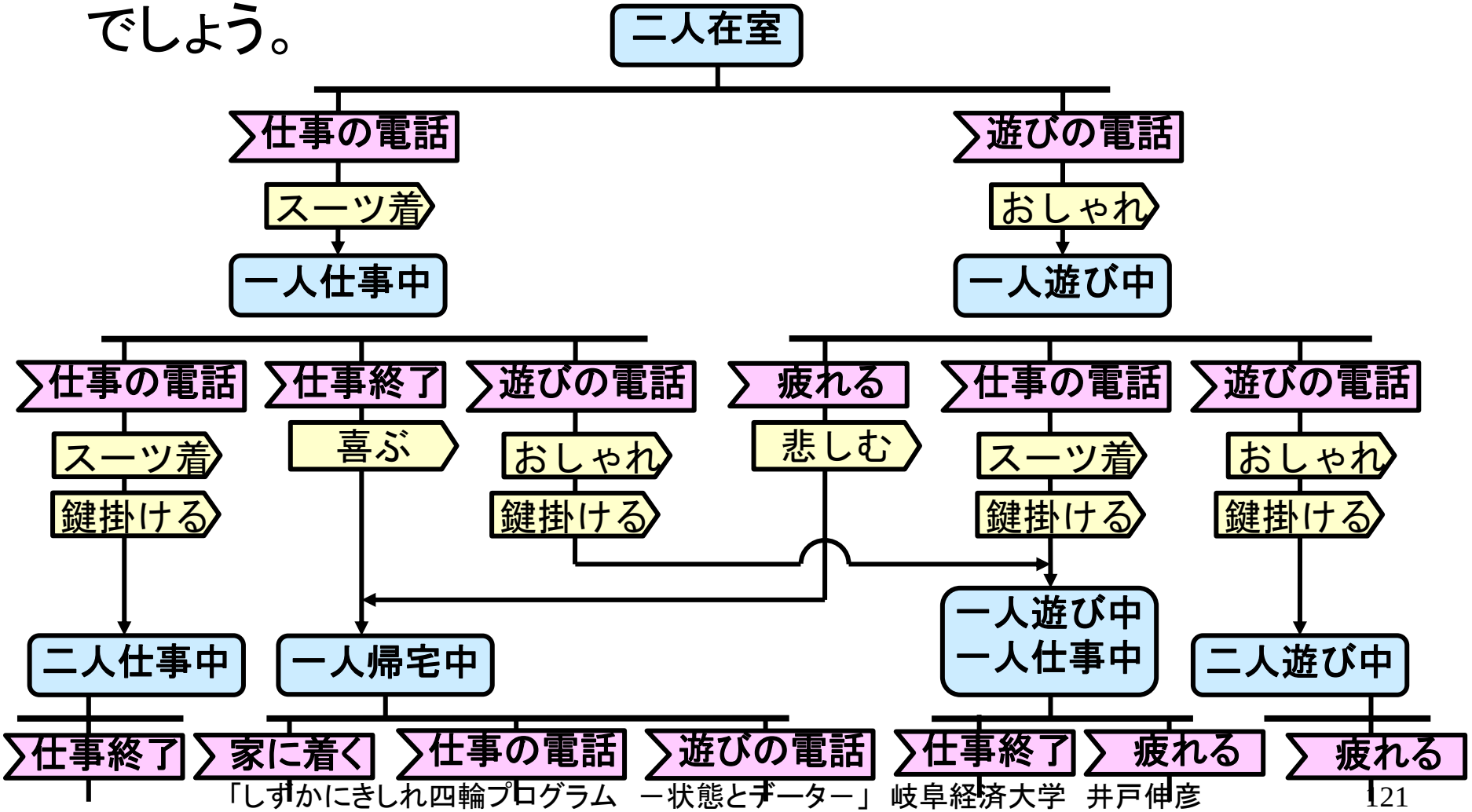
（６．２．１）ひとりの状態遷移図

■スライド(6.2)に示したルームメイトの、“一人”についての状態遷移を示します。

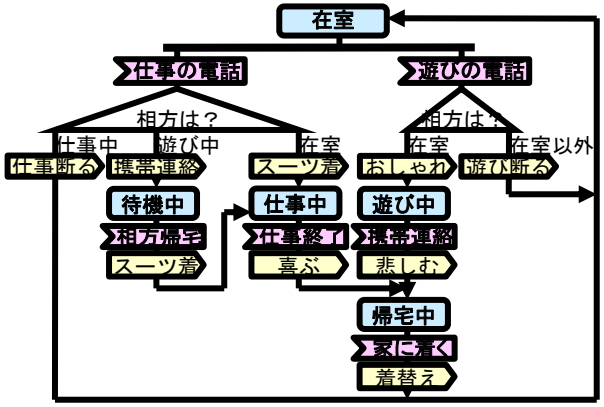
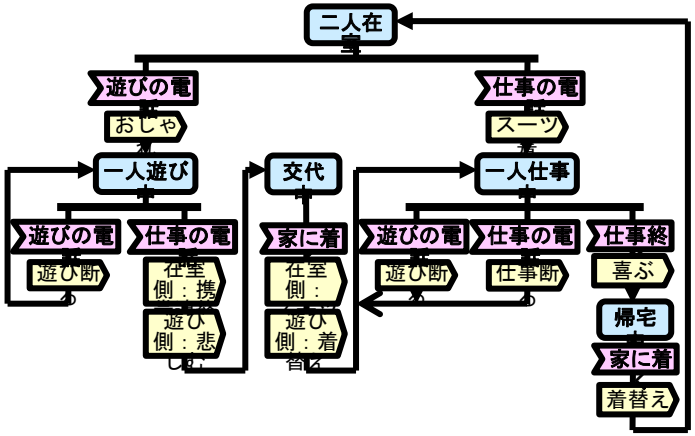
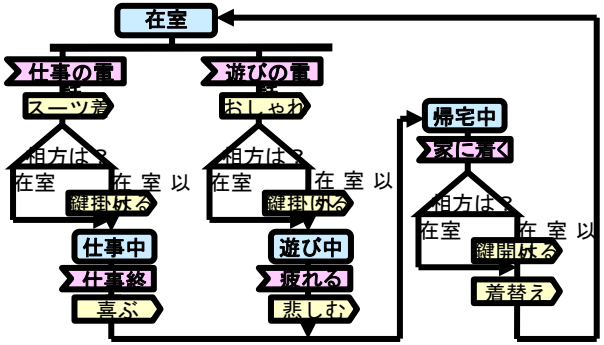
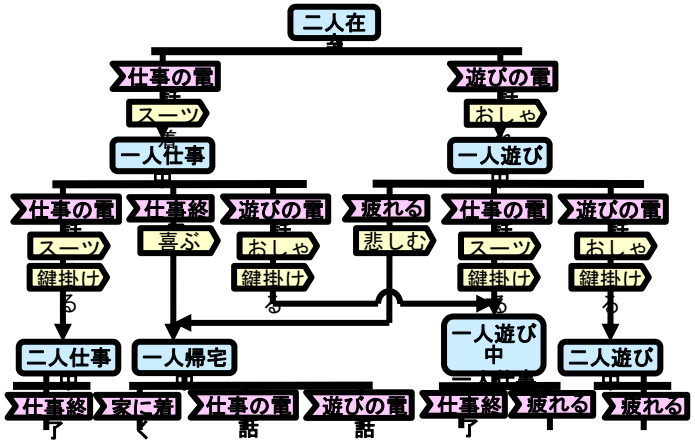


（6．2．2）二人の状態遷移図

■状態の数が増えすぎて、スライド1枚では書ききれません。今度はスライド(6.2.1)のほうが良いことは明らかでしょう。



(6．3)整理すると。。。

	一人の状態遷移図	二人の状態遷移図
ルームメイト その1 密接 	(6.1.1) 	(6.1.2) 
ルームメイト その2 疎遠 	(6.2.1) 	(6.2.2) 

（6．3．1）“密接”と“疎遠”

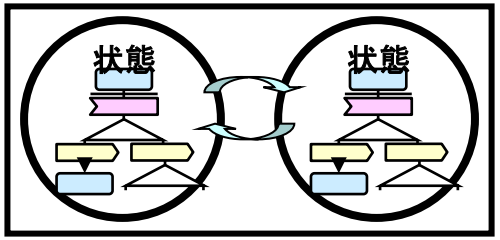
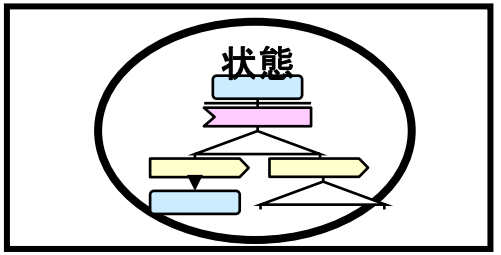
■スライド(6.1)と(6.2)とのルームメイトを比較します。
- (6.1)は、相方の動きに強く依存した生活をしています。
- (6.2)は、鍵の開け閉め以外は互いに関係の無い独立した生活をしています。

■スライド(6.2.1)では、独立した2つのものを一つの状態遷移図にまとめようとしたため、状態数が増えてしまいました。

	在室	仕事中	遊び中	帰宅中
在室	二人在室中	一人仕事中	一人遊び中	一人帰宅中
仕事中		二人仕事中	一人仕事中 一人遊び中	一人仕事中 一人帰宅中
遊び中			二人遊び中	一人遊び中 一人帰宅中
帰宅中				二人帰宅中

（6．3．2）状態をまとめること

■状態をまとめる場合の得失を考えます。

	複数の状態遷移 	まとめた状態遷移 
状態遷移図中の状態の数	○：少ない	×：多い
2つの状態遷移図での相互作用	×：煩雑	○：状態遷移図内に含まれる
全体で起こることの解りやすさ	×：2つの状態の組合せを考える必要あり	○：状態遷移図にそのまま現れる
個別で起こることの解りやすさ	○：状態遷移図にそのまま現れる	×：状態遷移図中から判断する必要がある

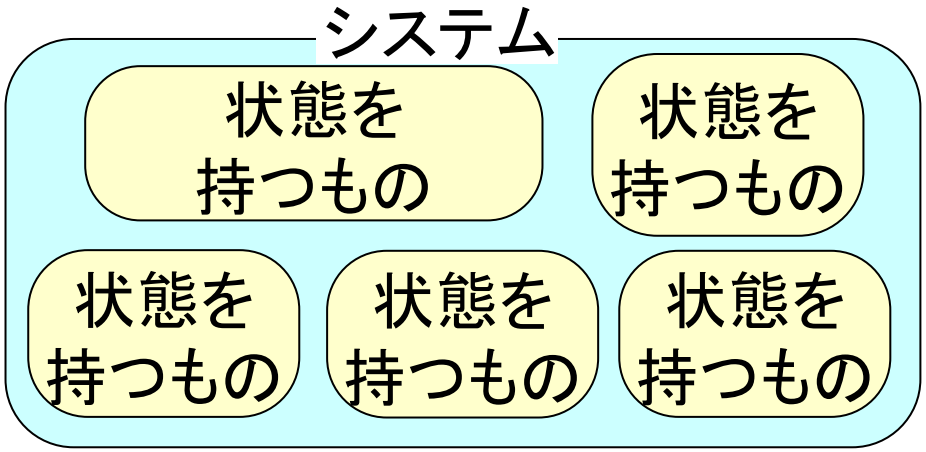
（６．４）一般的な設計の問題

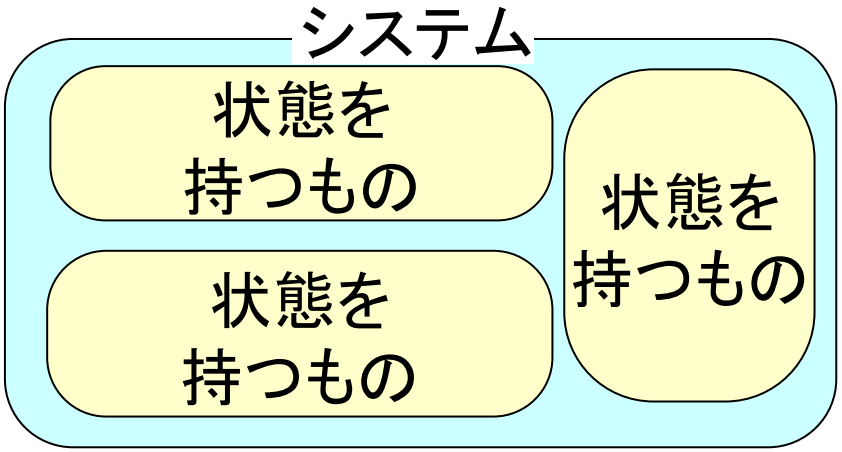
■ここまで（６）節では、「同じ状態遷移をもつ２つのもの」を見てきました。

■システムを設計する上では、「ことなる状態遷移をもつ２つのもの」も、当然存在します。これらを２つのままにしておくか、１つにまとめるかも、設計上の問題となる訳です。

■さらに一般的に言えば、「システムの中に状態を持つものをどのように配置するか」という問題となります。この課題は、オブジェクト指向設計とも密接に関係します。

（課題：どのように状態をもつものを配置するか？）





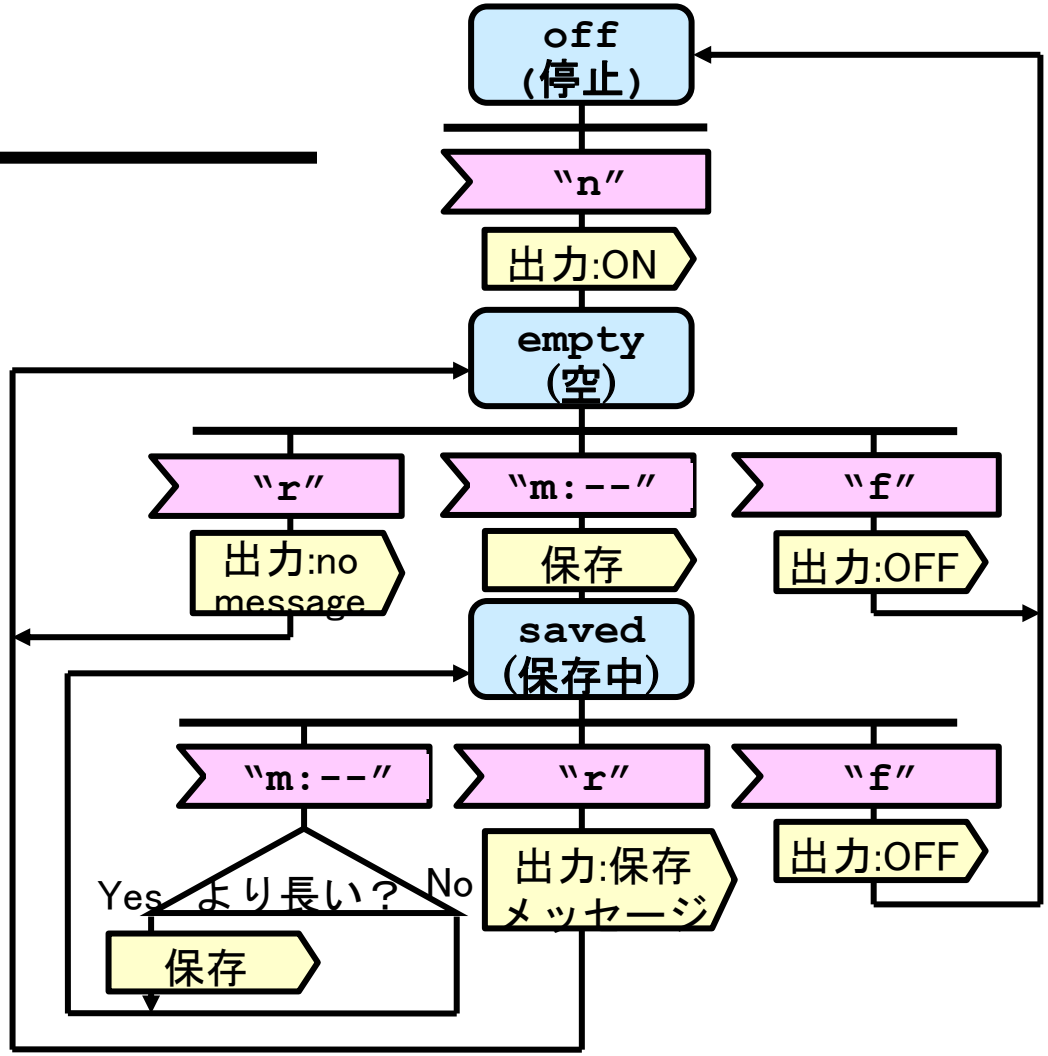
（6．5）演習（課題6－1）

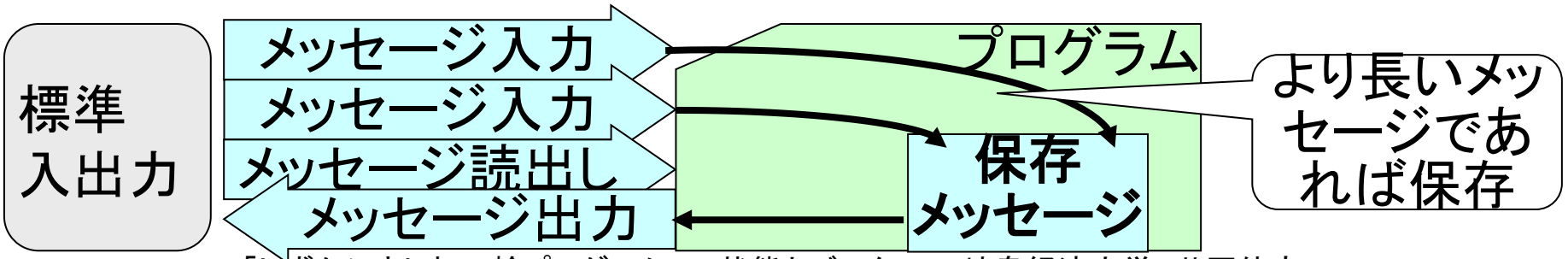
■次の状態のペアのうち、（一般的に考えて）より密接な関係があるものを選んでください。

- （1）（A)大阪の天気と東京の天気、（B)大阪の天気と大垣の天気
- （2）（A)ピッチャーの状態とキャッチャーの状態、
  （B)ピッチャーの状態と外野手の状態
- （3）（A)対戦ゲームでの二人の競技者の状態、
  （B)ロールプレイングゲームでの二人の競技者の状態
- （4）（A)オーディオの、アンプの状態とCDプレイヤーの状態、
  （B)オーディオの、CDプレイヤーの状態と
  ラジオチューナーの状態

(7)状態の階層化

- スライド(5.3)で考えた、メッセージ保存について再度取り上げます。
- 個々では、メッセージの保存は、ハードディスクに行うこととします。

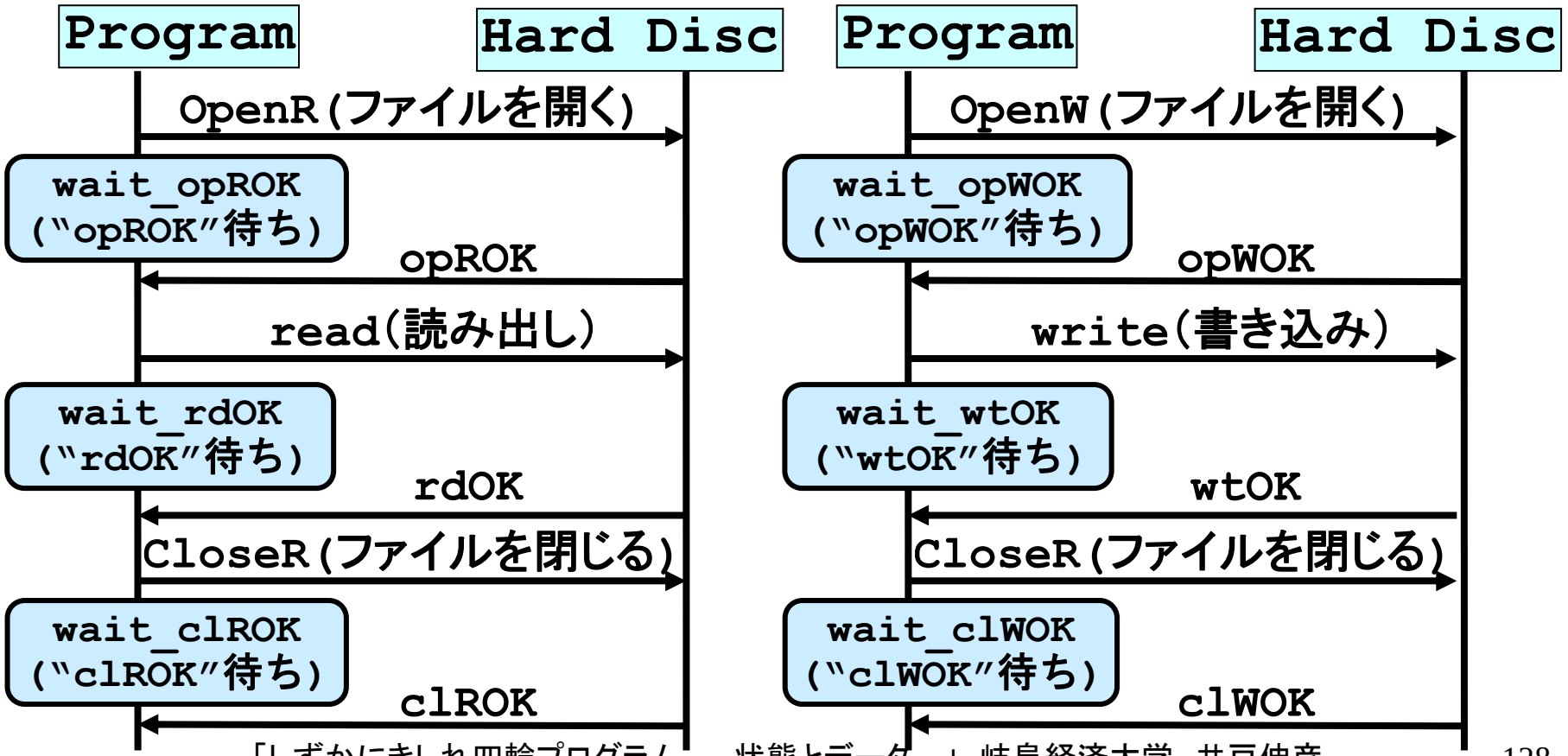




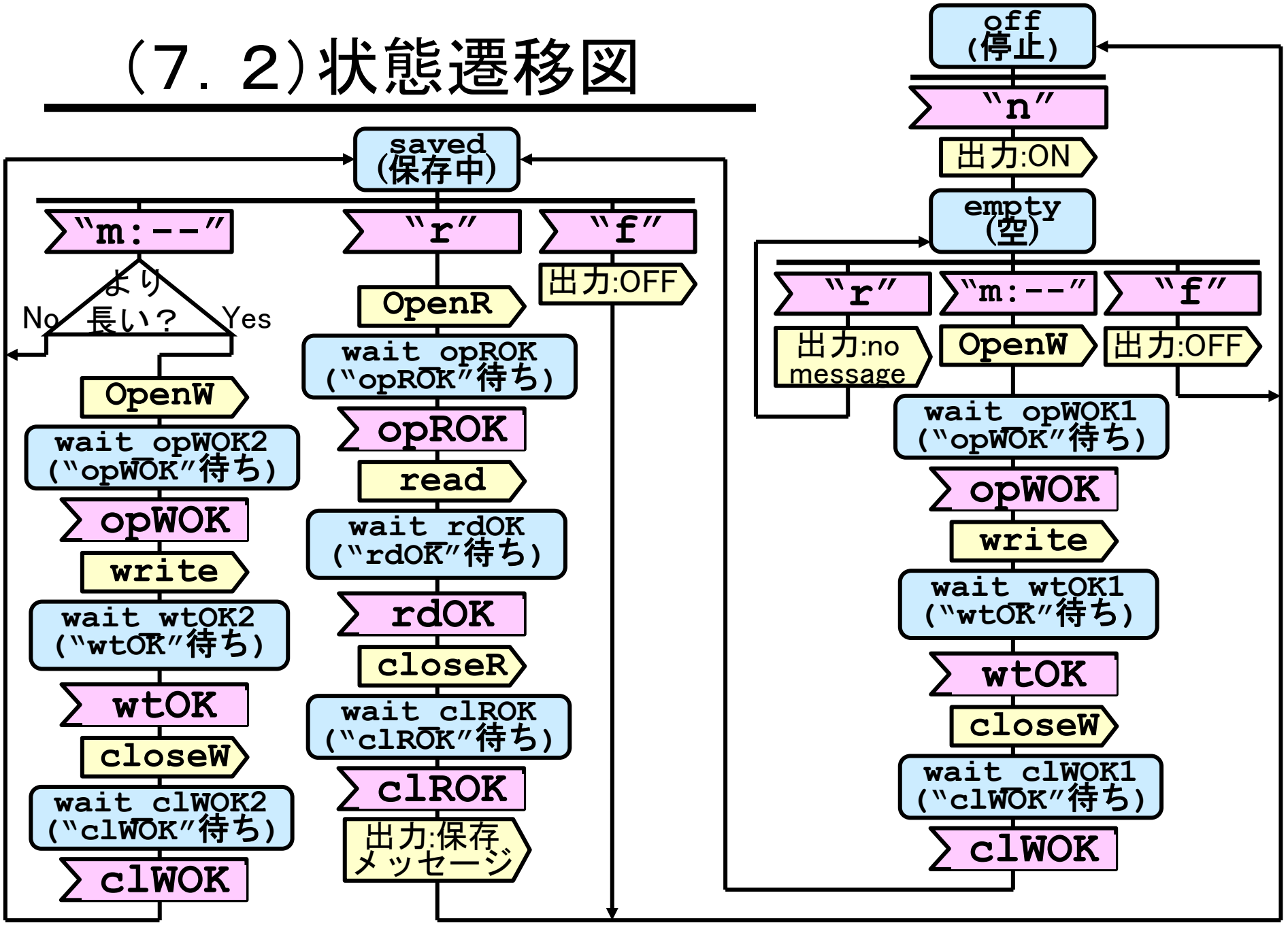
（7．1）ハードディスクへの保存

■ハードディスクへの読み出し・書き込みは、次のようなシーケンスで行うものとします。

■これをスライド(7)の遷移図に追加すると、少々ややこしいのですが、次スライド(7.2)の状態遷移図を得ます。



(7．2)状態遷移図

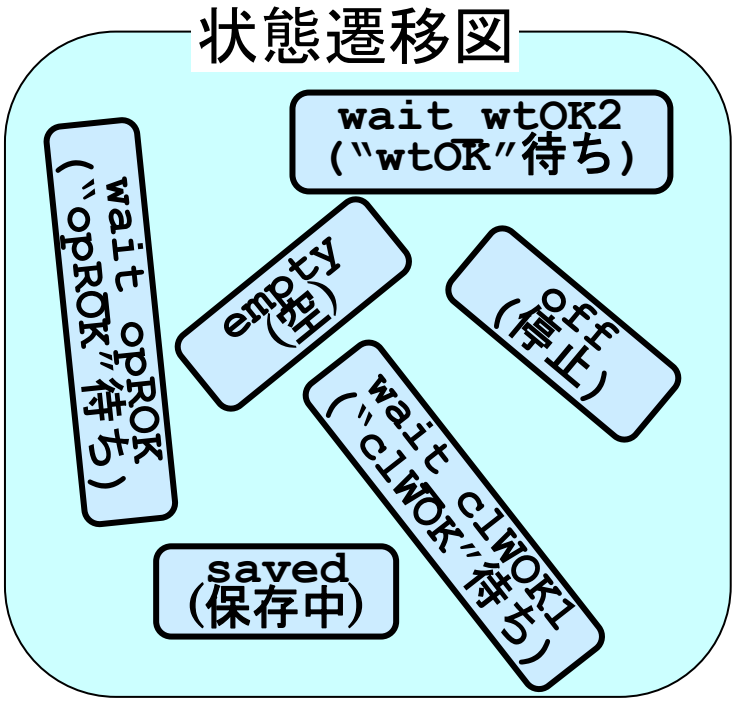

off
(停止)
"n"
出力:ON
empty
(空)
"r"
"m:--"
"f"
出力:no message
OpenW
出力:OFF
wait opWOK1
("opWOK"待ち)
opWOK
write
wait wtOK1
("wtOK"待ち)
wtOK
closeW
wait clWOK1
("clWOK"待ち)
clWOK
saved
(保存中)
"m:--"
"r"
"f"
出力:OFF
より
No 長い？ Yes
OpenW
wait opWOK2
("opWOK"待ち)
opWOK
write
wait wtOK2
("wtOK"待ち)
wtOK
closeW
wait clWOK2
("clWOK"待ち)
clWOK
OpenR
wait opROK
("opROK"待ち)
opROK
read
wait rdOK
("rdOK"待ち)
rdOK
closeR
wait clROK
("clROK"待ち)
clROK
出力:保存メッセージ

（7．3）これでいいのか？

■前スライド(7.2)の状態遷移図は間違いではありません。このとおりにプログラムを作っても、ちゃんと動作します。

■しかしながら、次のような問題点により、解りにくくなっています。

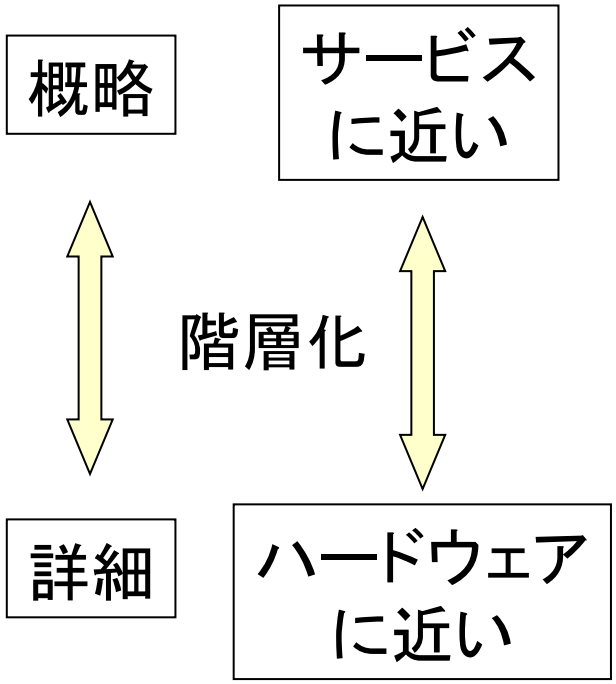
- メッセージに関することと、ハードディスクにアクセスすることとが、同じ遷移図の中に描かれており、煩雑である。
- 状態の数が多く、わかり難い。

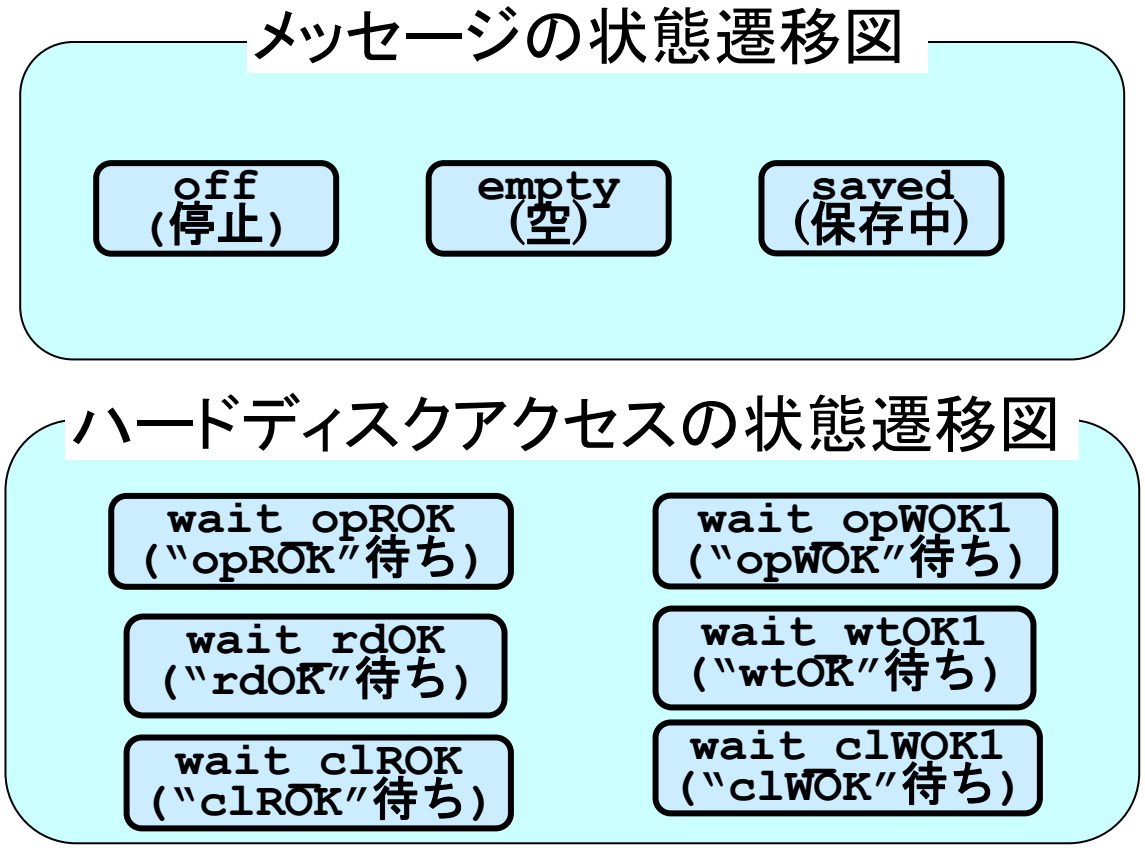


（7．4）階層化

■「サービスに近いか、ハードウェアに近いか」の基準により分類すると、次の2つは分けて考えることが妥当です。

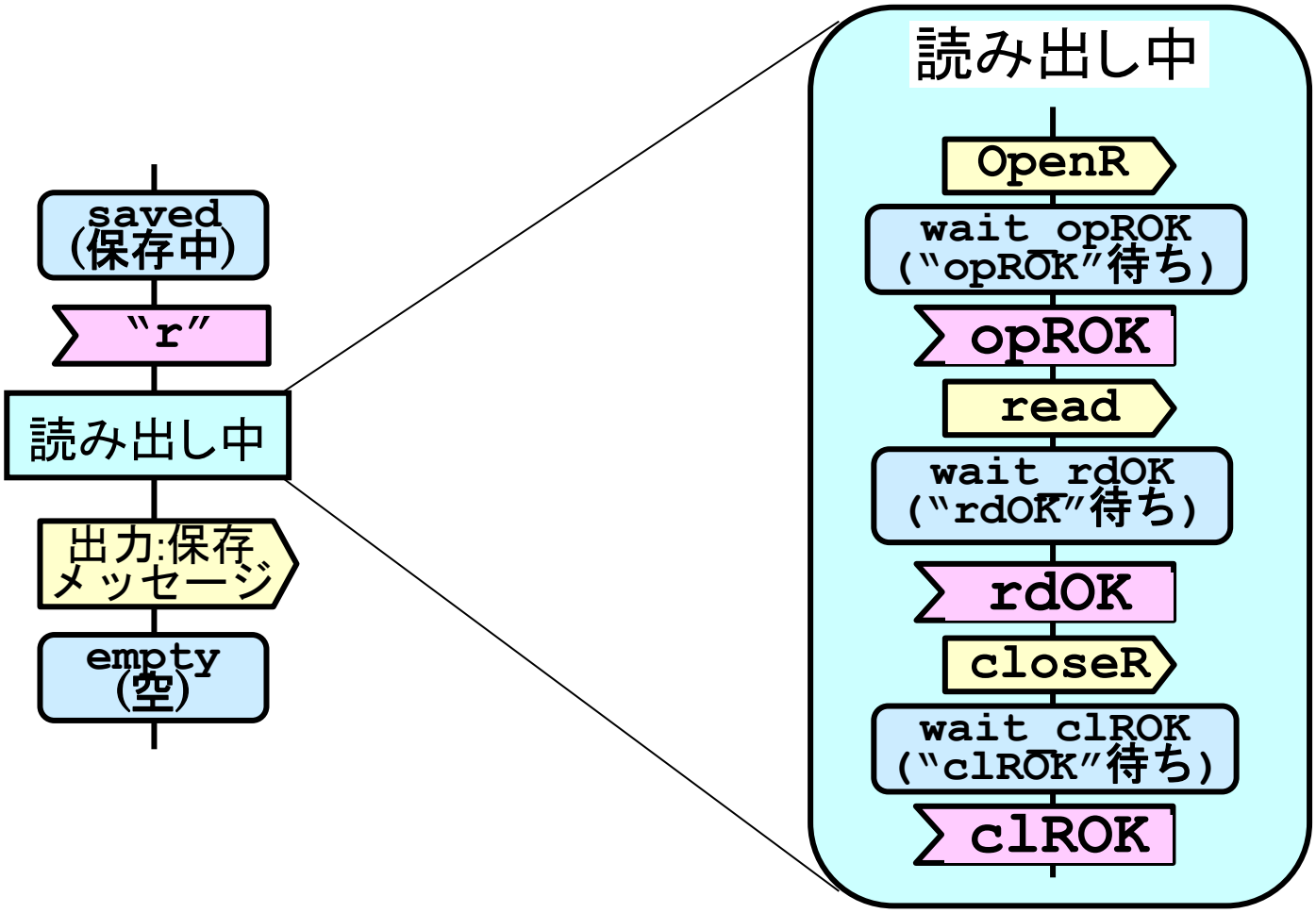
- スライド(5.3.4)：メッセージの保存に関する状態遷移
- スライド(7.1)：ハードディスクへのアクセスに関する状態遷移





（７．５）詳細な部分を概略にまとめる

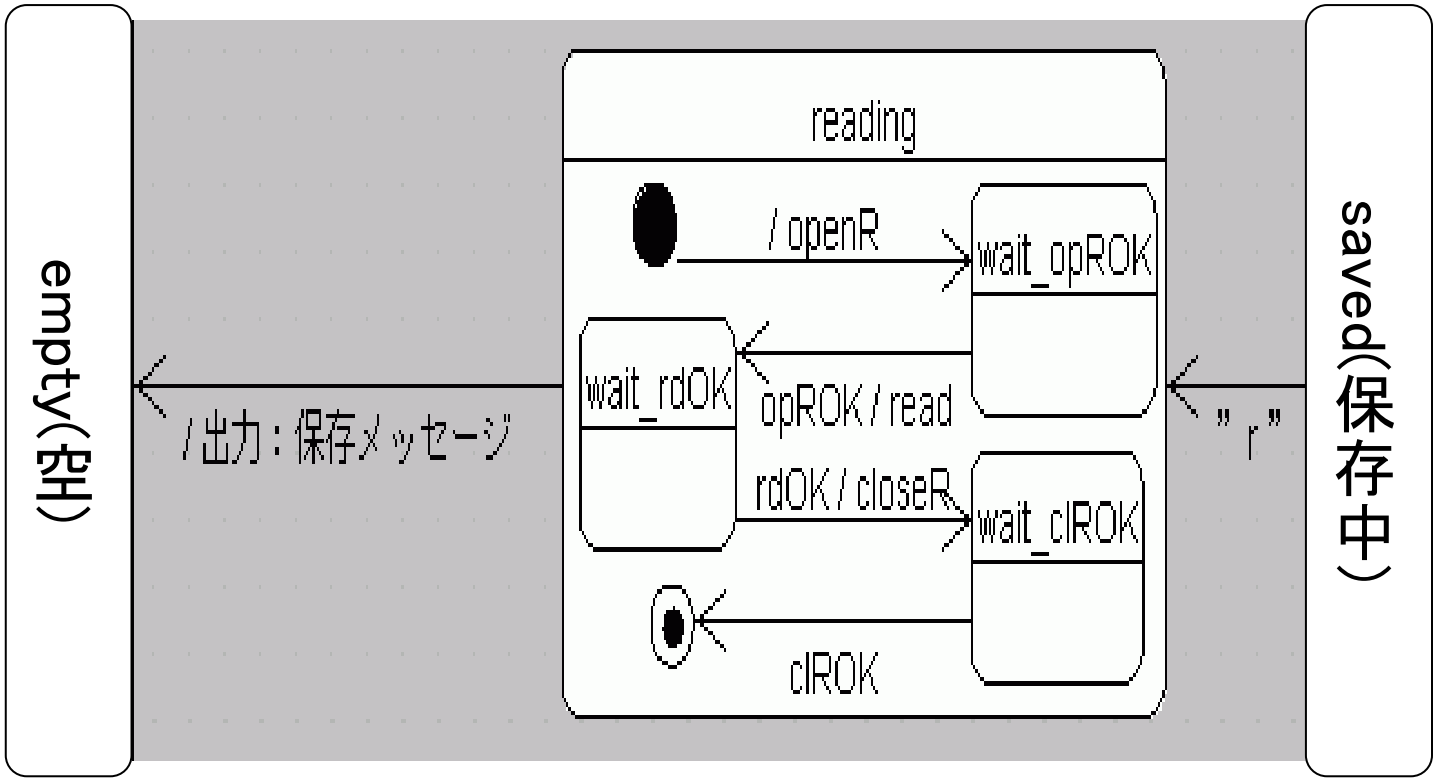
■概略であるメッセージのレベルでの状態遷移図では、詳細であるハードディスクのレベルでの状態遷移図を記さず、まとめておくことにすれば、遷移図はすっきりします。

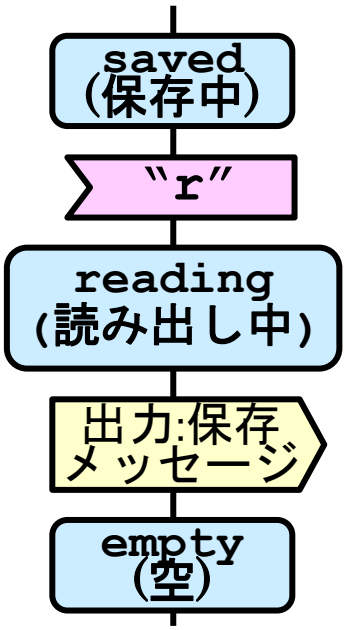


（7．6）複合状態(Composite State)

■UMLでは、階層のことなる状態遷移を一つの図に書き込む記法があります。

■下図のように、状態の中に状態遷移を書き込んでいきます。このように内部に状態遷移を持つ状態を、複合状態と呼びます。

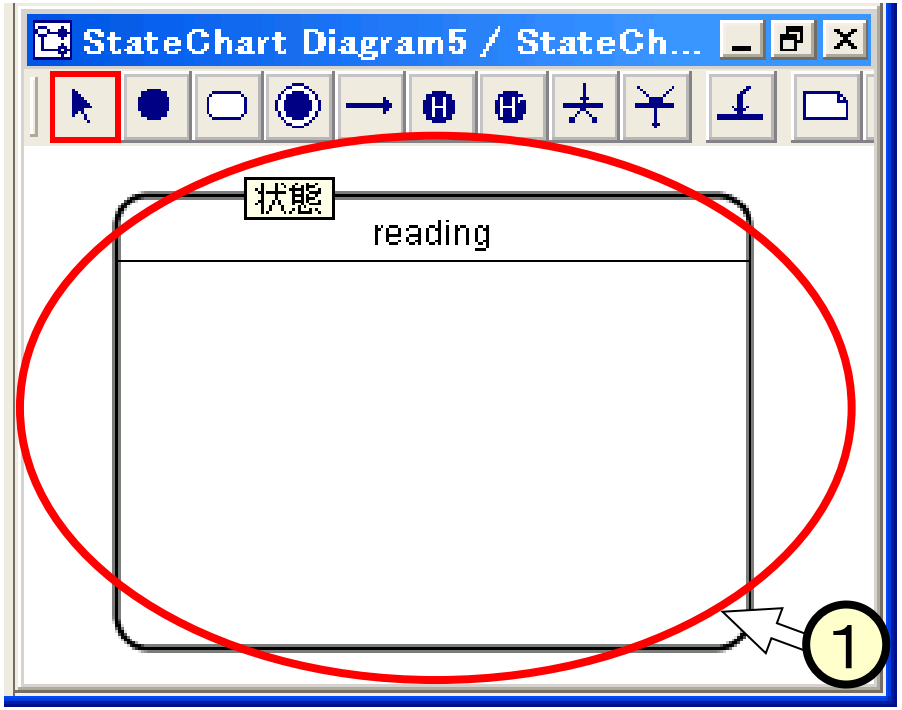


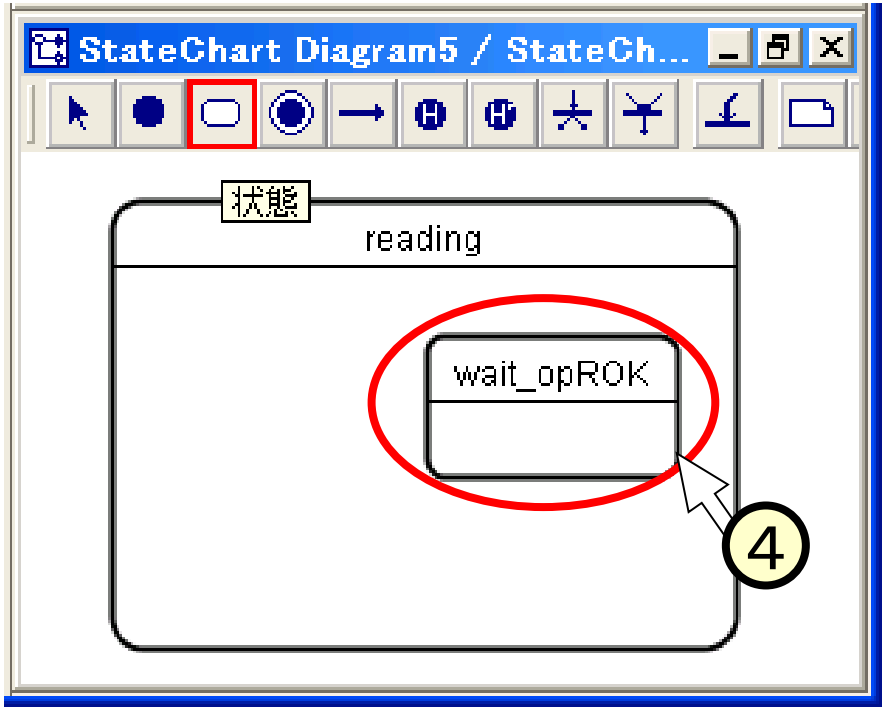


（7．7．1）複合状態の書き方　－1－

■複合状態とする状態を作成し、これを大きくしておきます（①）。

■描いた複合状態の内部には、通常と同じ手順で、状態を書き加える（②）ことが出来ます。

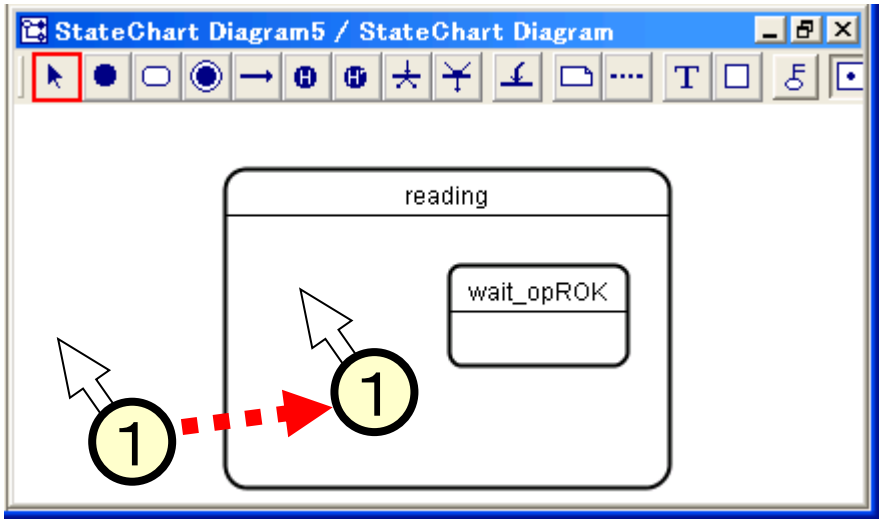


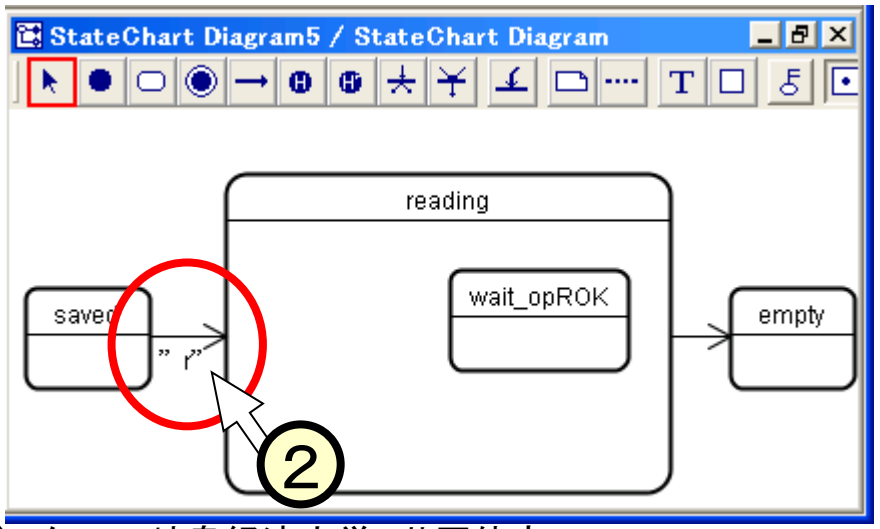


（7．7．2）複合状態の書き方　－2－

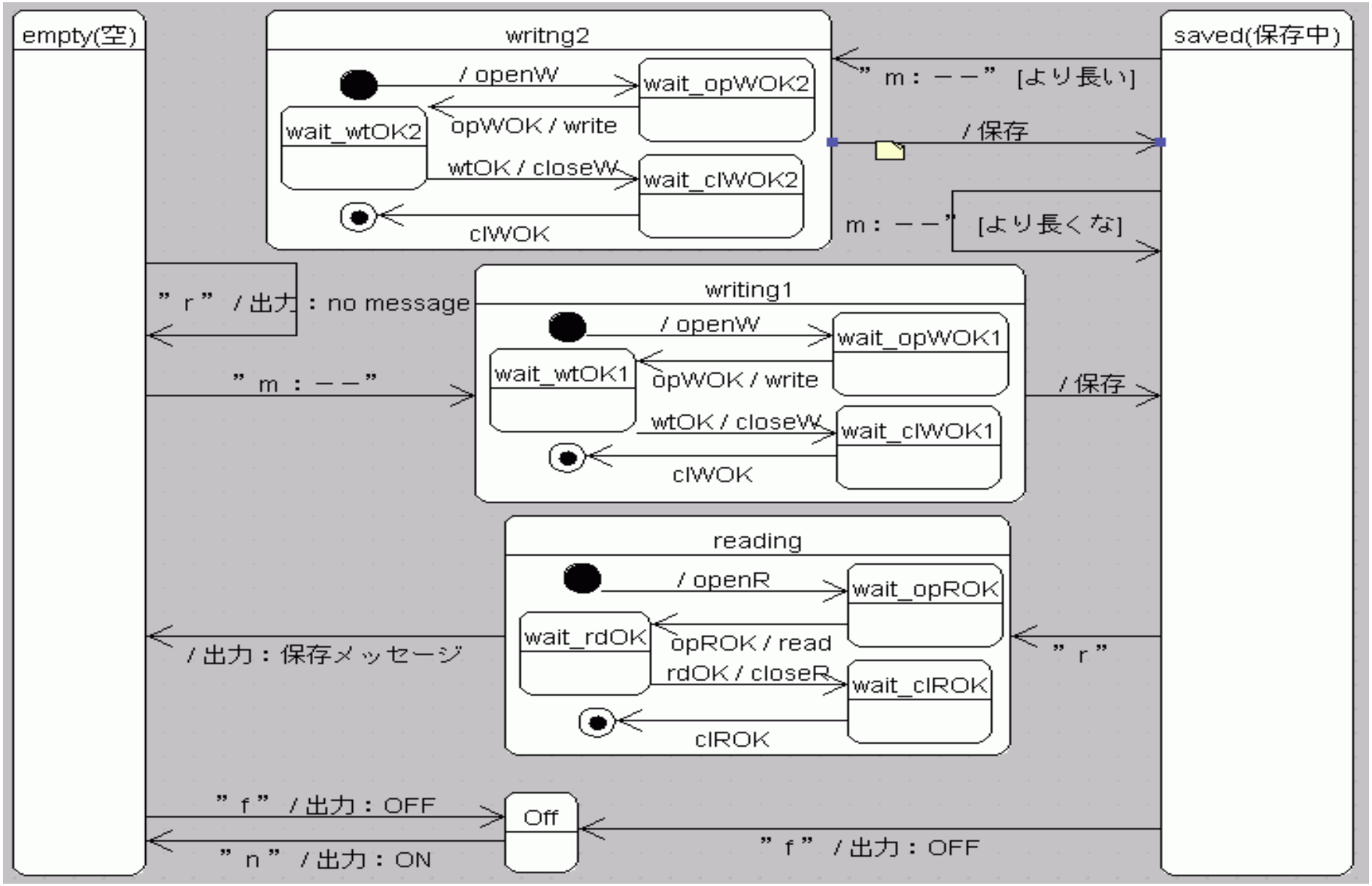
■複合状態をドラッグ（①）すると、内部の状態と一緒に移動します。

■複合状態と状態との遷移要素（②）は、状態と状態との遷移と同様の方法で描くことが出来ます。





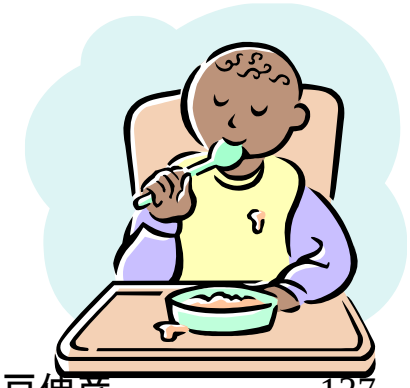
（7．8）複合状態を用いた状態遷移図



（7．9）演習（課題7－1）

■次の事実を、複合状態を用いた状態遷移図で描いてください。

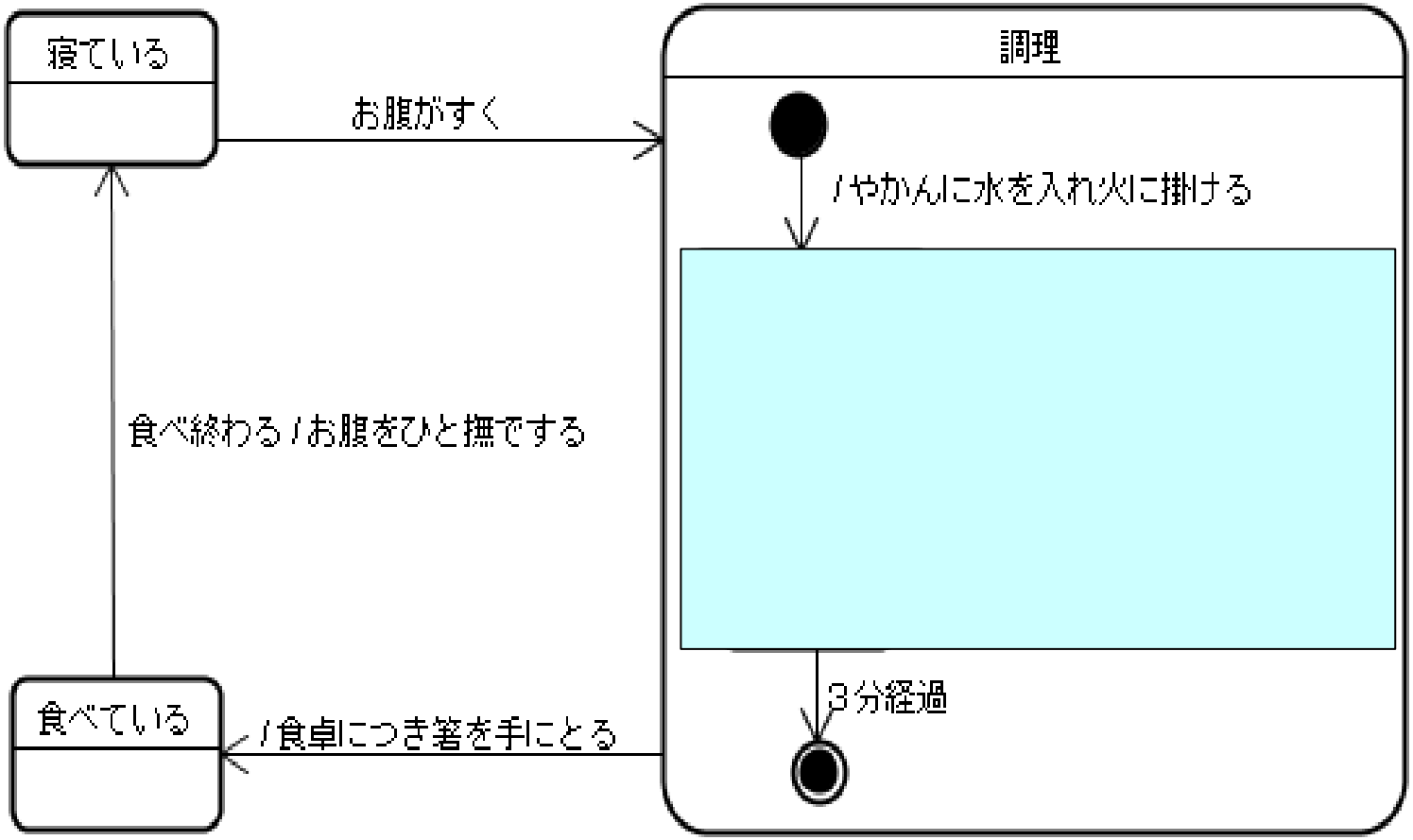
- （1）休日、A君は、(A)寝ているか、（B）調理しているか、(C)食べているかのいずれかの状態にある。
  - ◆(A)の状態で、お腹がすくと、(B)に移る。
  - ◆(B)は複合状態であり、下記（2）の手続きを経て(C)に移る。
  - ◆(C)の状態で、食べ終わると、(A)に移る。
- （2）(B)調理については、次の手続きを経て、(C)に移る。
  - ◆やかんに水を入れ火にかける。
  - ◆湯が沸騰したら、カップラーメンに湯を注ぐ。
  - ◆3分経ったら、食卓につき、箸を手にとる。



■次のスライドにヒントを示します。

（7．10）課題7－1ヒント

■下の未完の状態遷移図に遷移要素を付け加えて完成させてください。



（8．1）情報データ

■この（8）節では、“情報データ”（スライド(1.3)）を扱います。

■思考実験

- スライド(1.2.1)で扱ったデータについて考えてみます。

□方法A

名前	伊藤
所属	陸上
住所	市内

名前	佐藤
所属	陸上
住所	市外

名前	田中
所属	水泳
住所	県外

名前	鈴木
所属	野球
住所	市外

名前	斉藤
所属	野球
住所	県外

名前	中村
所属	水泳
住所	市内

□方法B

〈陸上〉
伊藤
佐藤

〈水泳〉
田中
中村

〈野球〉
鈴木
斉藤

〈市内〉
伊藤
中村

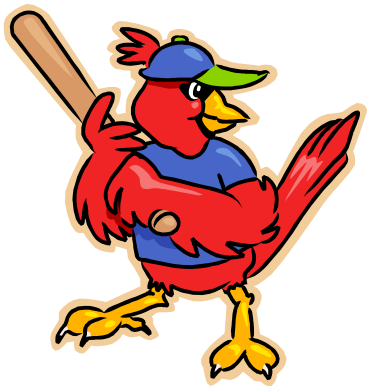
〈市外〉
佐藤
鈴木

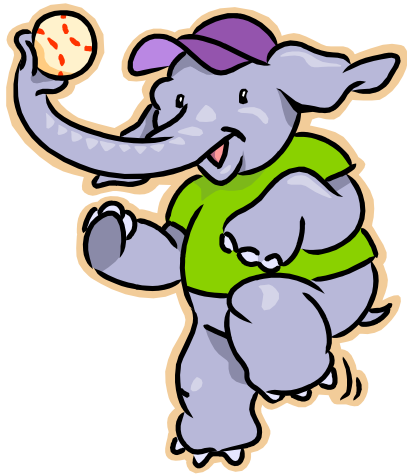
〈県外〉
田中
斉藤

（8．2）問題を解く

■次の問題について、スライド(8.1)の2つの方法での解法を考えます。

- 問題1：野球部の学生の名前の一覧を挙げてください。
- 問題2：野球部の学生が住んでいる住所（市内・市外・県外）を挙げてください。

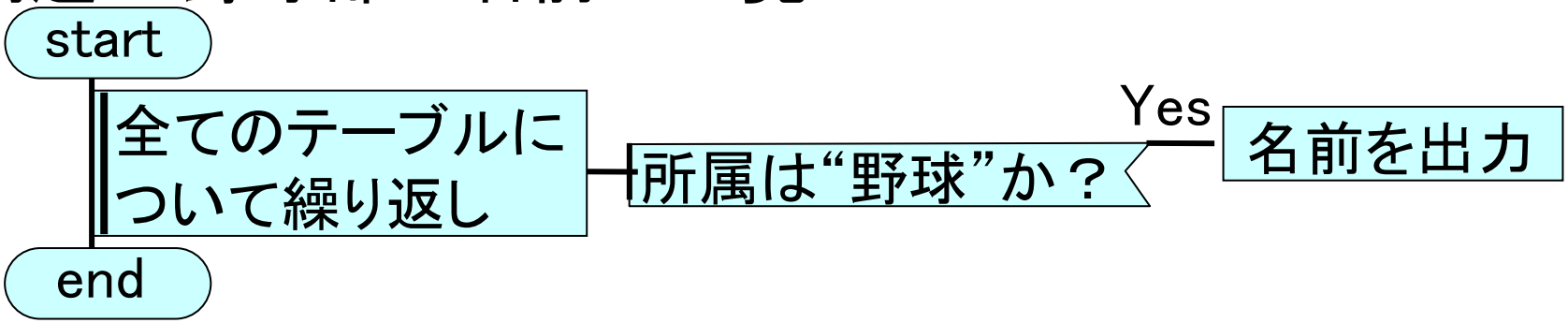




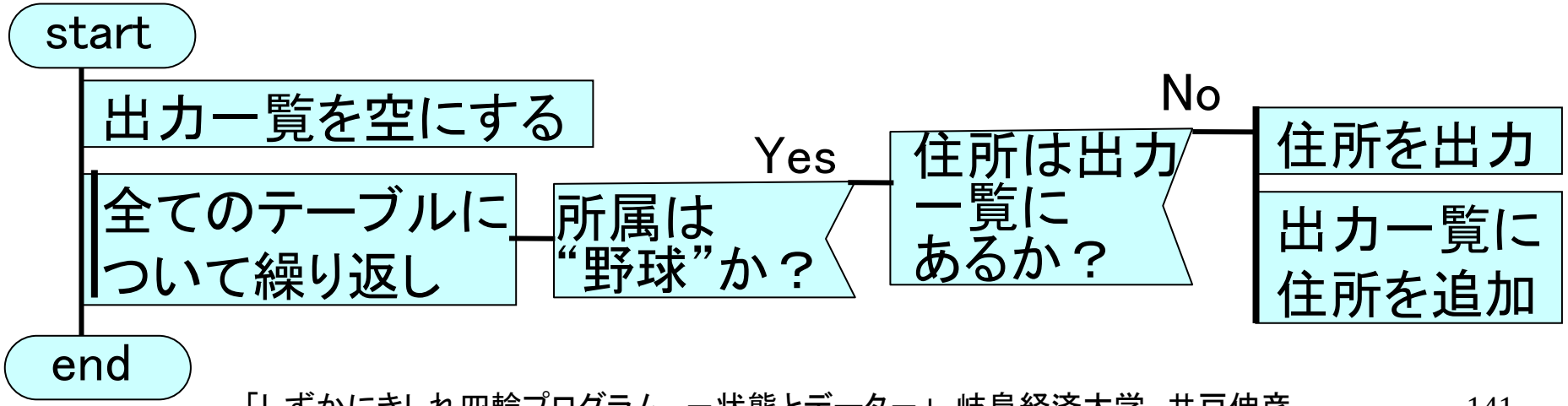
（8．2．1）方法Aでの処理

名前	伊藤	名前	佐藤	名前	田中	名前	鈴木	名前	斉藤	名前	中村
所属	陸上	所属	陸上	所属	水泳	所属	野球	所属	野球	所属	水泳
住所	市内	住所	市外	住所	県外	住所	市外	住所	県外	住所	市内

■問題１：野球部の名前の一覧



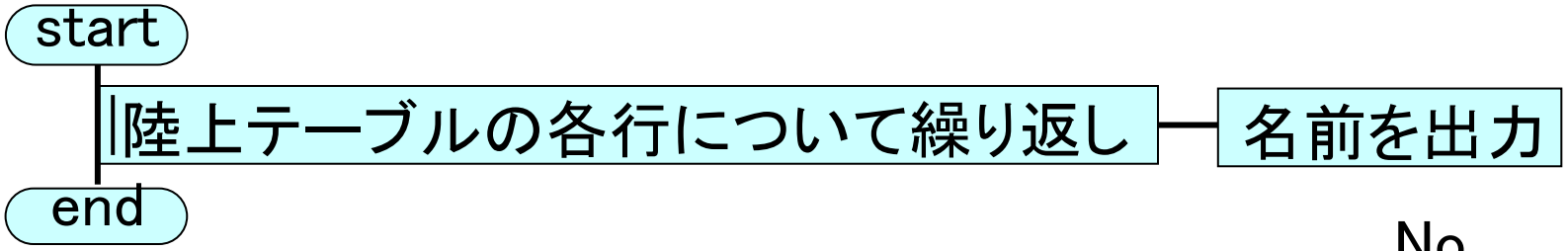
■問題２：野球部の住所一覧



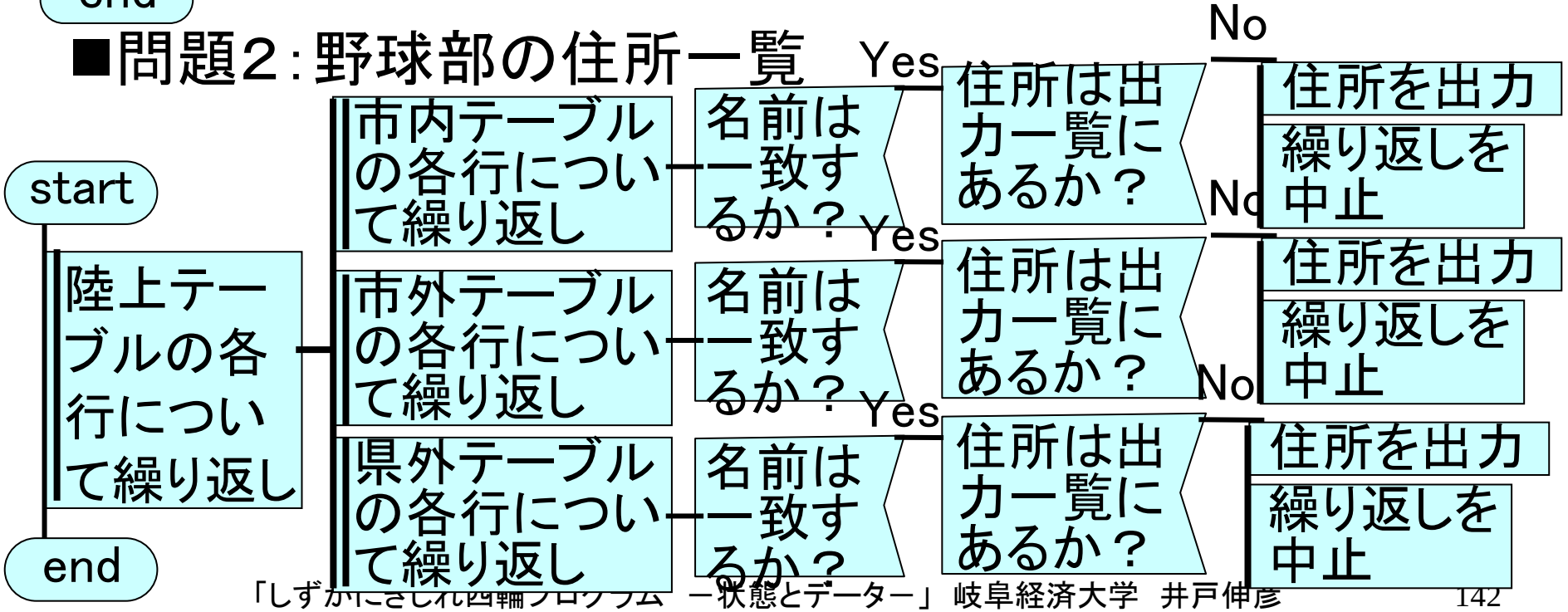
（8．2．2）方法Bでの処理

〈陸上〉	〈水泳〉	〈野球〉	〈市内〉	〈市外〉	〈県外〉
伊藤	田中	鈴木	伊藤	佐藤	田中
佐藤	中村	斉藤	中村	鈴木	斉藤

■問題１：野球部の名前の一覧



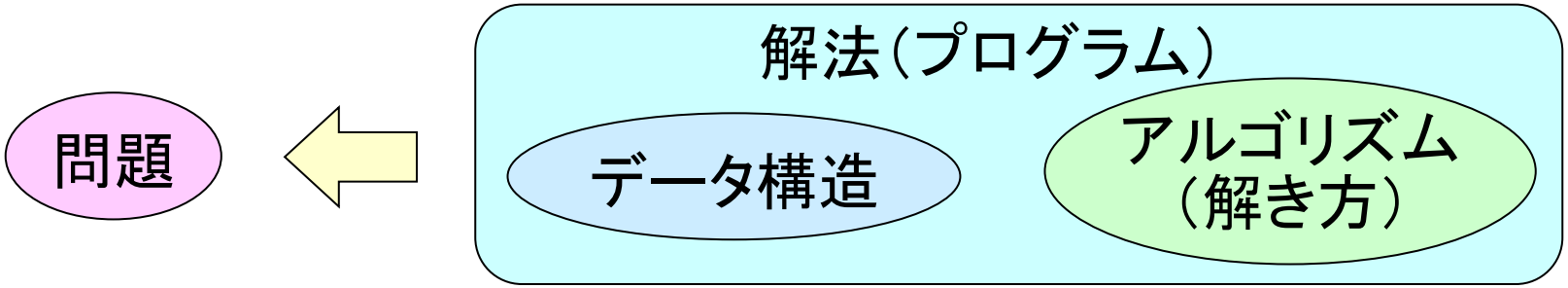
■問題２：野球部の住所一覧



（8．2．3）テーブルの構成

■スライド(8.2.1),(8.2.2)で見たように、テーブルの構成（＝データ構造とも言います）により、解き方（＝アルゴリズムとも言います）が変わってきます。

■ある問題に対して、これの解法（プログラム）は、“データ構造”と“アルゴリズム”から成り立つわけです。



■類型化された問題に対して、どのような解法がより効率が良いかについては、様々な研究があります。

- 類型化された問題：ex. 複数の数を大きさ順に並べよ。
  http://www.ics.kagoshima-u.ac.jp/~fuchida/edu/algorithm/sort-algorithm/
- 「アルゴリズム＋データ構造＝プログラム」、Niklaus Wirth、1979

（8．2．4）私たちのプログラム

■私たちがプログラムを作る際には、“類型的な問題”を解く訳ではありません（ですから、アルゴリズムというような言い方はあまりしません）。

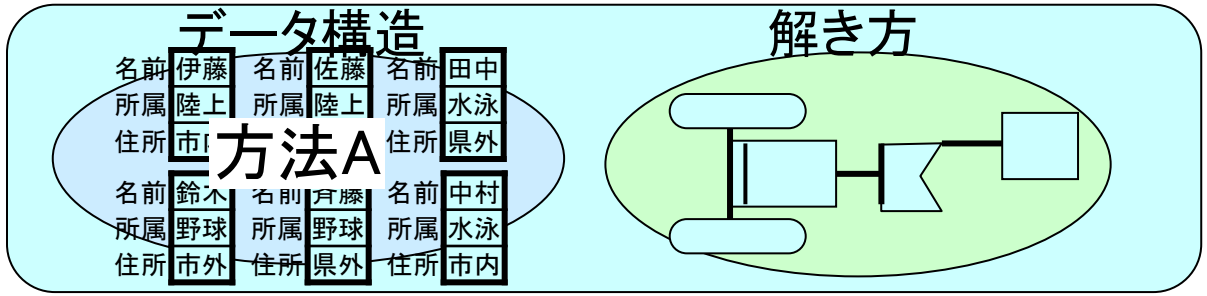
■しかしながら、データ構造を決めることが、プログラム設計の重要な部分であることには変わりありません。

■例えば、スライド(8.2)の問題に対して、スライド(8.1)の方法A,Bのいずれを採るかにより、次のスライドに示すように、解き方は異なってきました。

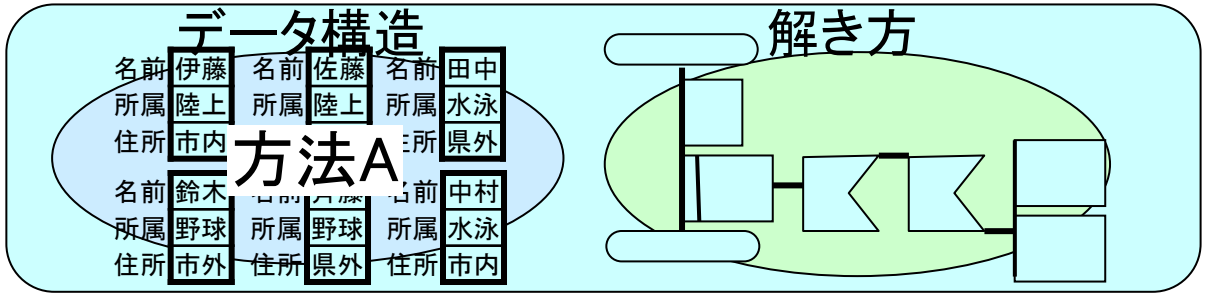
- 問題1には、方法Bのほうが方法Aよりも簡単です。
- 問題2には、方法Aのほうが方法Bよりも簡単です。

（8．2．5）問題とデータ構造・解き方

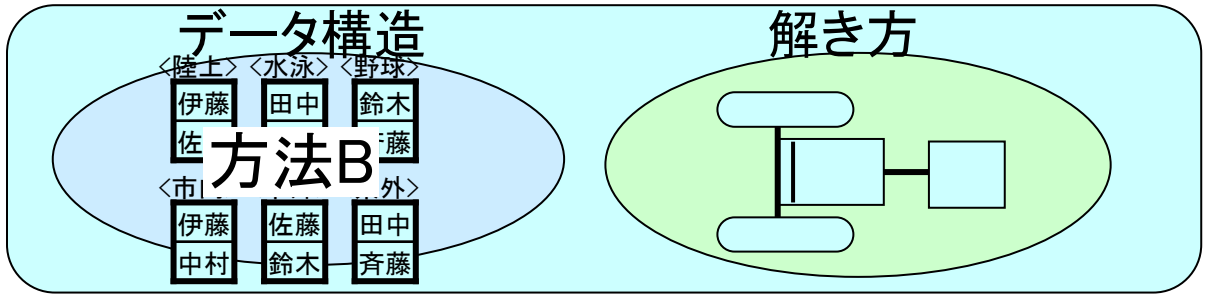
問題1：
野球部の
名前の一覧



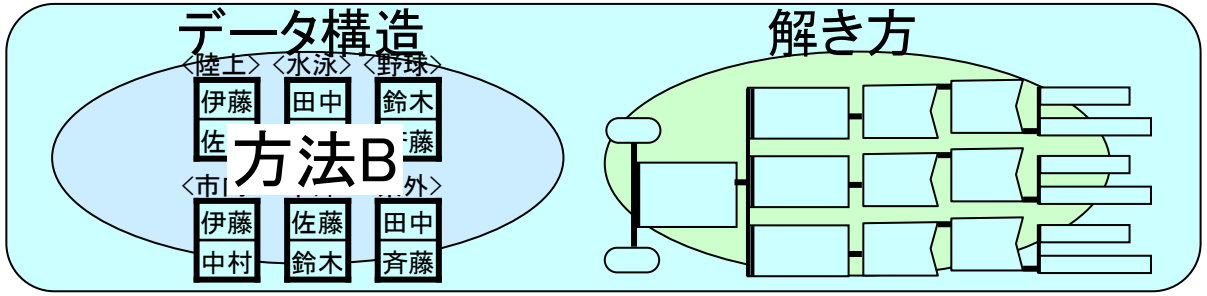
問題2：
野球部の
住所の一覧



問題1：
野球部の
名前の一覧



問題2：
野球部の
住所の一覧

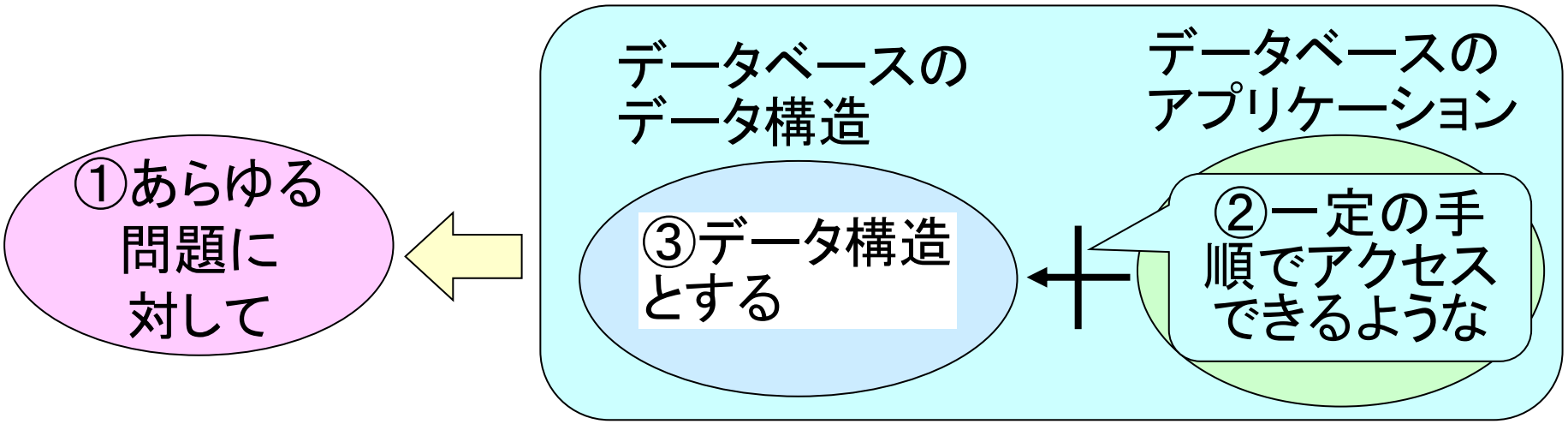


（8．2．6）データベースのデータ構造

■少し本題から逸れますが、データベースのデータ構造について説明します。

■データベースでは、将来どのような問題が設定された場合でも、一定の手順でデータが利用出来るようなデータ構造を基本としています。

■よって、特定の問題によらない考え方（例えば、リレーショナルモデル）により、データ構造を決める訳です。



（8．3．1）インデックス

■ここまで考えてきたデータには、インデックスは使われていませんでした。

■インデックスを使うと、テーブルの各行についてひとつひとつ調べることなく、目的のデータを取り出すことが出来ます。

■例：鈴木さんの部活の顧問を調べる

インデックスなし

・テーブル1から、
鈴木さんの所属を探す。
・テーブル2から、
野球部の顧問を探して
出力する。

〈テーブル1〉

名前	所属
伊藤	陸上
田中	水泳
鈴木	野球

〈テーブル2〉

部活	顧問
陸上	森
水泳	堀
野球	南

インデックスあり

・テーブル1から、
鈴木さんの所属のインデックスを
探す。
・テーブル2より、
インデックス位置の顧問を
出力する。

〈テーブル1〉

名前	所属(i)
伊藤	1
田中	2
鈴木	3

〈テーブル2〉

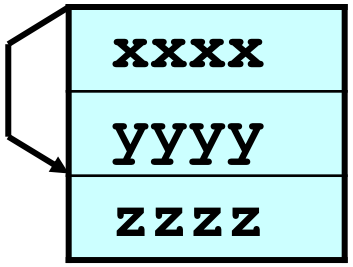
i	部活	顧問
1	陸上	森
2	水泳	堀
3	野球	南

（8．3．2）配列、ポインタ、ハッシュ

■詳細には触れませんが、前スライド(8.3.1)にて、“インデックス”と呼んだものは、具体的には配列やポインタなどで実現されます。

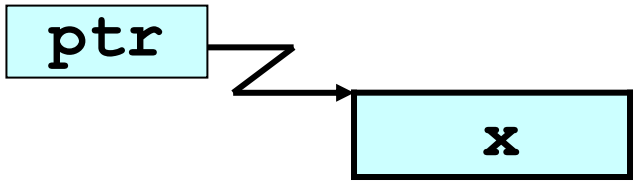
- 配列

```
string aa[] = {
      “xxxx”,
      “yyyy”,
      “zzzz”};
```



- ポインタ

```
char bb = ‘x’;
char *ptr = &bb;
```



■“ハッシュ”という言葉を聞いたことがありますか？ハッシュもインデックスする方法のひとつです。

（8．4．1）インデックスを作ってみる

■下図のようなテーブルで表されるデータについて、インデックスを用いたテーブルを作成してみることにします。

＜＜電気製品問屋のデータ＞＞

＜顧客＞

顧客番号	顧客名
C1	A商店
C2	Bマート
C3	Cストア

＜商品＞

商品番号	商品名	価格
G1	テレビ	198,000
G2	洗濯機	59,000

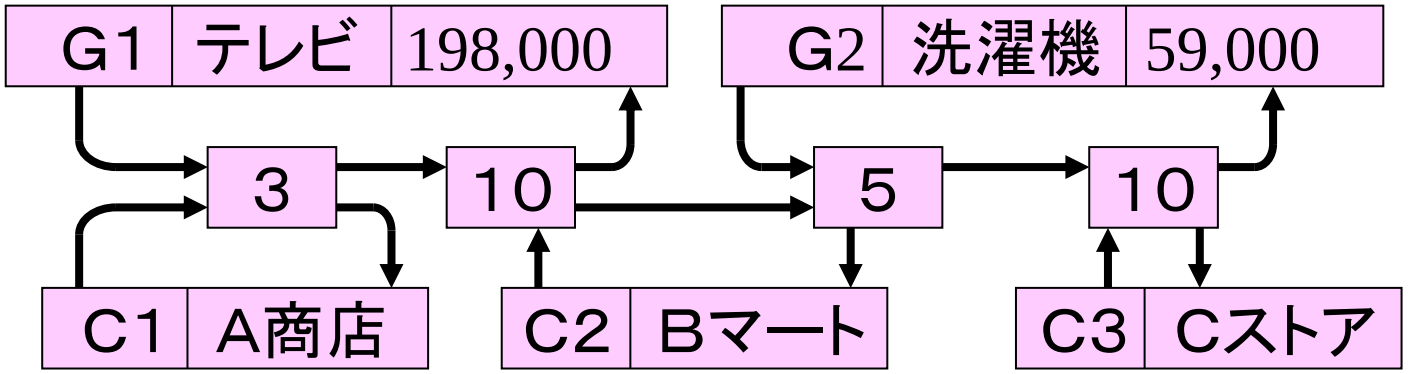
＜納品＞

商品番号	顧客番号	納品数量
G1	C1	3
G1	C2	10
G2	C2	5
G2	C3	10

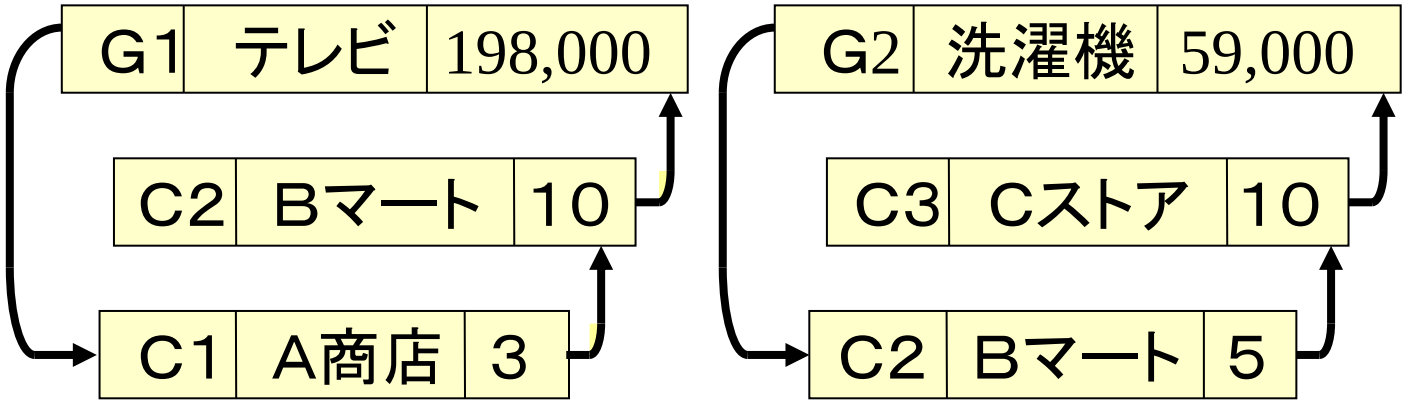
（8．4．2）様々なインデックスの方法

■前スライド(8.4.1)と同じ内容が表されていることを確認してください。

■例１



■例２

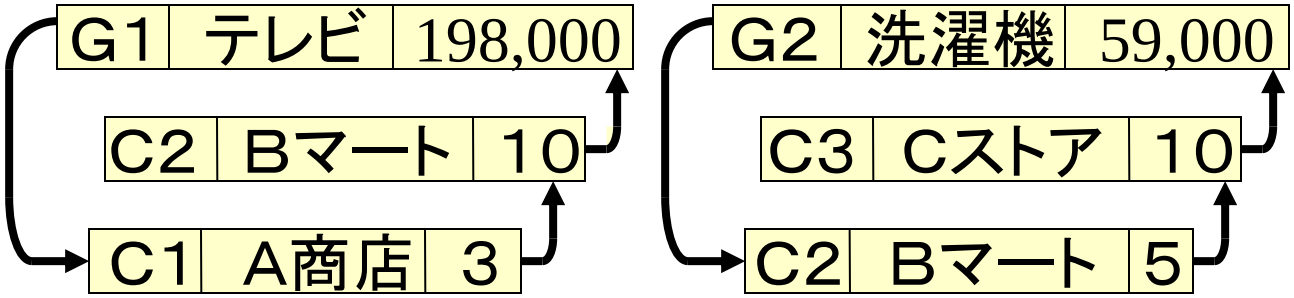


（ ⟶ :ポインタ）

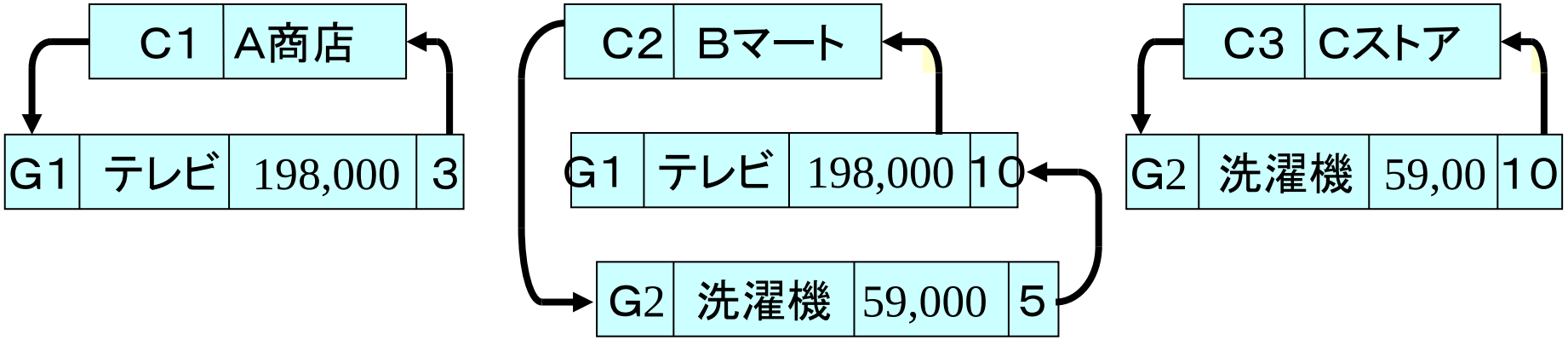
（8．4．3）目的による違い

■どのようにインデックスを構成するかにより、得意となる処理が出てきます。

- 例2（前スライドと同じ）：テレビの売上を調べたい



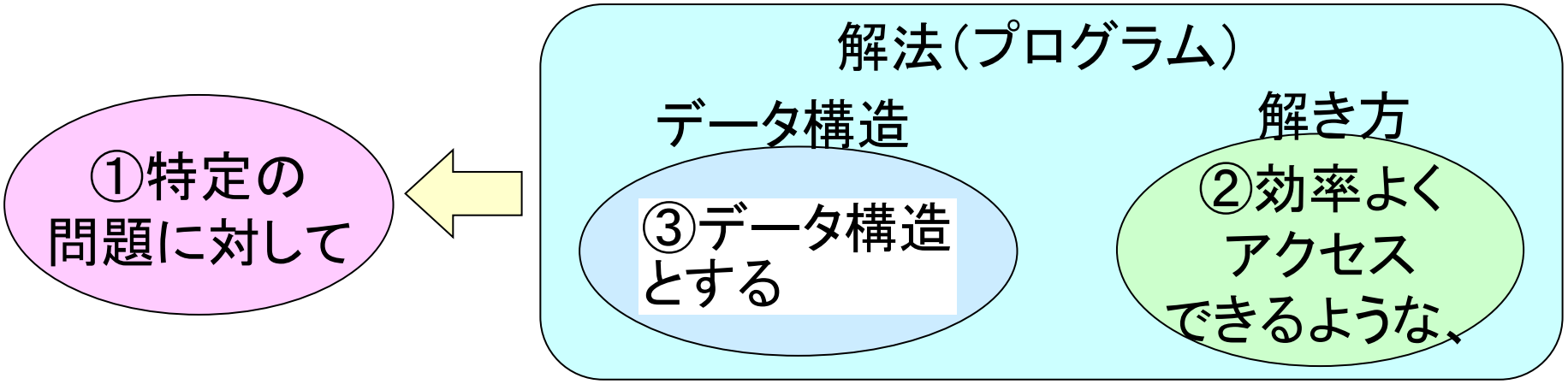
- 例3：Bマートへの売上を調べたい



（8．4．4）問題に沿ったデータ構造

■スライド(8.4.5)のように、特定の問題に合わせて、インデックス方法（＝データ構造）を考えていくことにより、より効率の良い解法（プログラム）が出来上がります。

■実際には、単に効率だけでなく、拡張性や了解性などの、他の観点にからも、データ構造の良し悪しは判断されます。



■スライド(8.2.6)に記したデータベースの場合は、あらゆる問題に対応することを主眼とするため、インデックスの使用を避けています。すなわち、スライド(8.4.1)のようなデータ構造を採る（リレーショナルモデルの場合）ことになります。

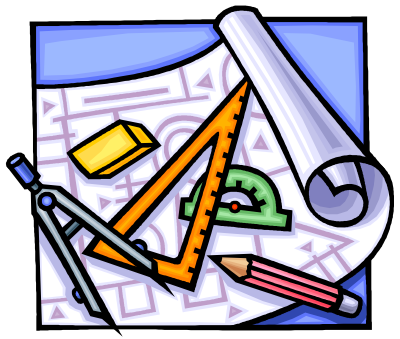
（8．5）オブジェクト指向

■昔であれば、プログラムを設計する際には、まず入念にデータ構造を設計しました。

■「ここまで説明してきたことを踏まえて、どのようにデータ構造を設計していくかについて考えていきます」と書きたいところですが、それは行いません。

■オブジェクト指向プログラミングでは、データ構造について、別の考え方をします。これについては、稿を改めて記していきます。

■筆者の考えでは、「オブジェクト指向であれば、データ構造に関する知見は要らない」というようなことは無いと考えます。オブジェクト指向においても、クラス構造の良し悪しは、データ構造と同じ問題を多く含むからです。



（８．６）演習（課題８－１）

■スライド（8.4.3)に記したインデックスの２つの例（例２、例３）から、次の問題を解くのに適したものを選んでください。

（１）売上金額トップの店を調べる。

（２）売上金額トップの製品を調べる。

（３）製品ごとの売上金額トップの店を調べる。

（４）店ごとの売上金額トップの製品を調べる。

（５）A商店よりテレビの売上金額の多い店を調べる。

（６）テレビより冷蔵庫の売上金額の多い店を調べる。