1.

INTRODUCTION TO COMPUTER SCIENCE MODELLING

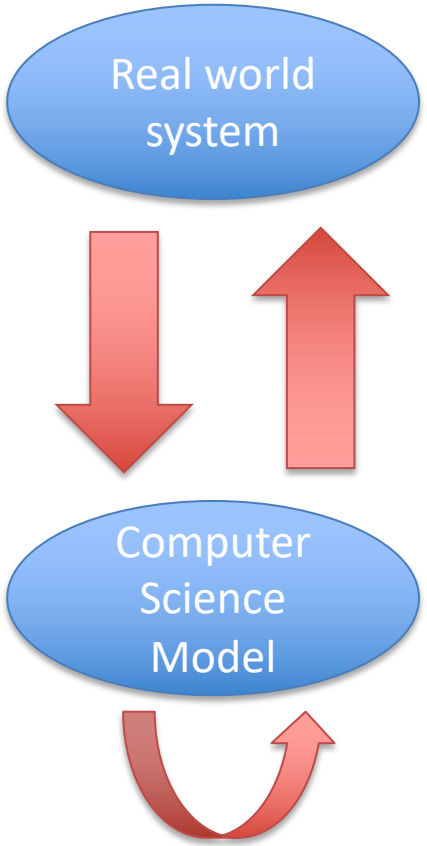
Overview of today's lecture

- Why study computer science models?
- Practical information
- Introduction to finite automata
- Alphabets, strings and languages
- By the way …
- Reading material and exercises

WHY STUDY COMPUTER SCIENCE MODELS?

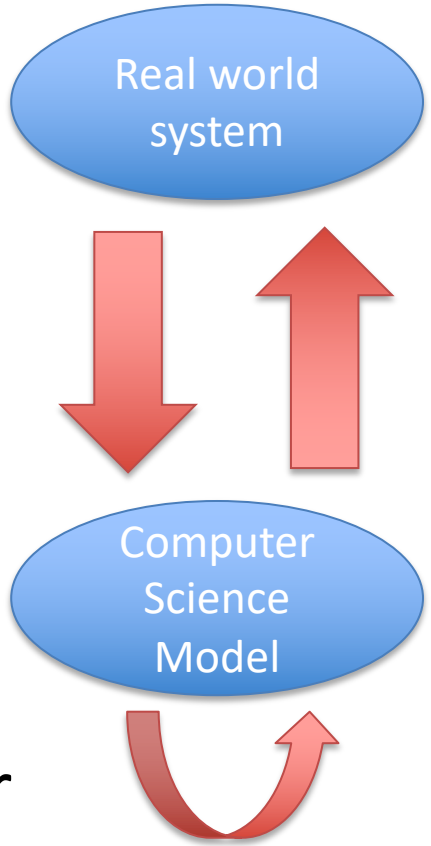
Computer Science models

- Computer science models describe some important real world behaviour
- They do so by abstracting, simplifying and codifying
- to the point where subtle analyses and observations and conclusions can be made
- and related back in a meaningful way to the physical world thereby shedding light on what was not obvious before



The role of CS models

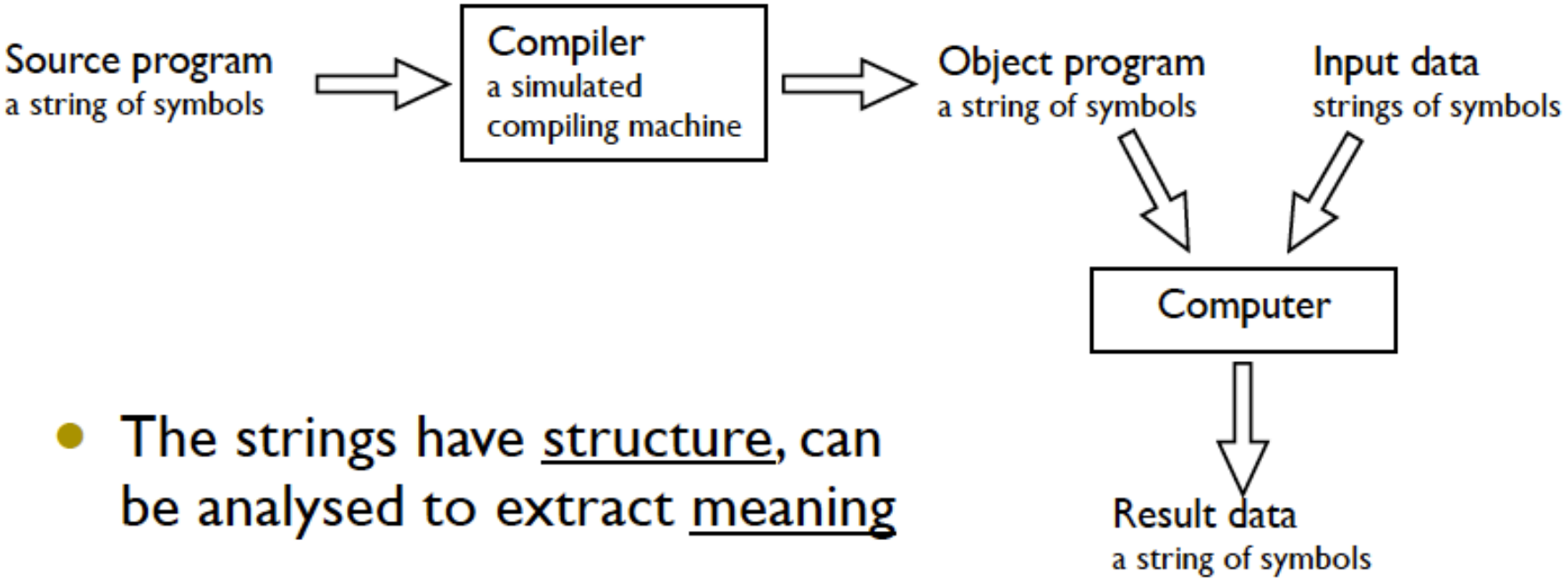
- To focus on selected aspects of systems/programs
- To structure the design of systems/programs
- To analyse the design of systems/programs
  - Correctness => safety by design
  - Security => privacy by design
  - Performance => energy saving by design
  - …
- Often the models give rise to tools for automatic solution for subsystems



Computer science models

- Added benefit: computer science models have many practical applications
  - Compilers: scanning, parsing, translation, …
  - Text search: editors, file systems, the web
  - Software verification: do programs behave as expected?
  - IDEs: model-driven software development
  - Security: enforcement of security policies
  - Programming techniques: table driven programming
  - …

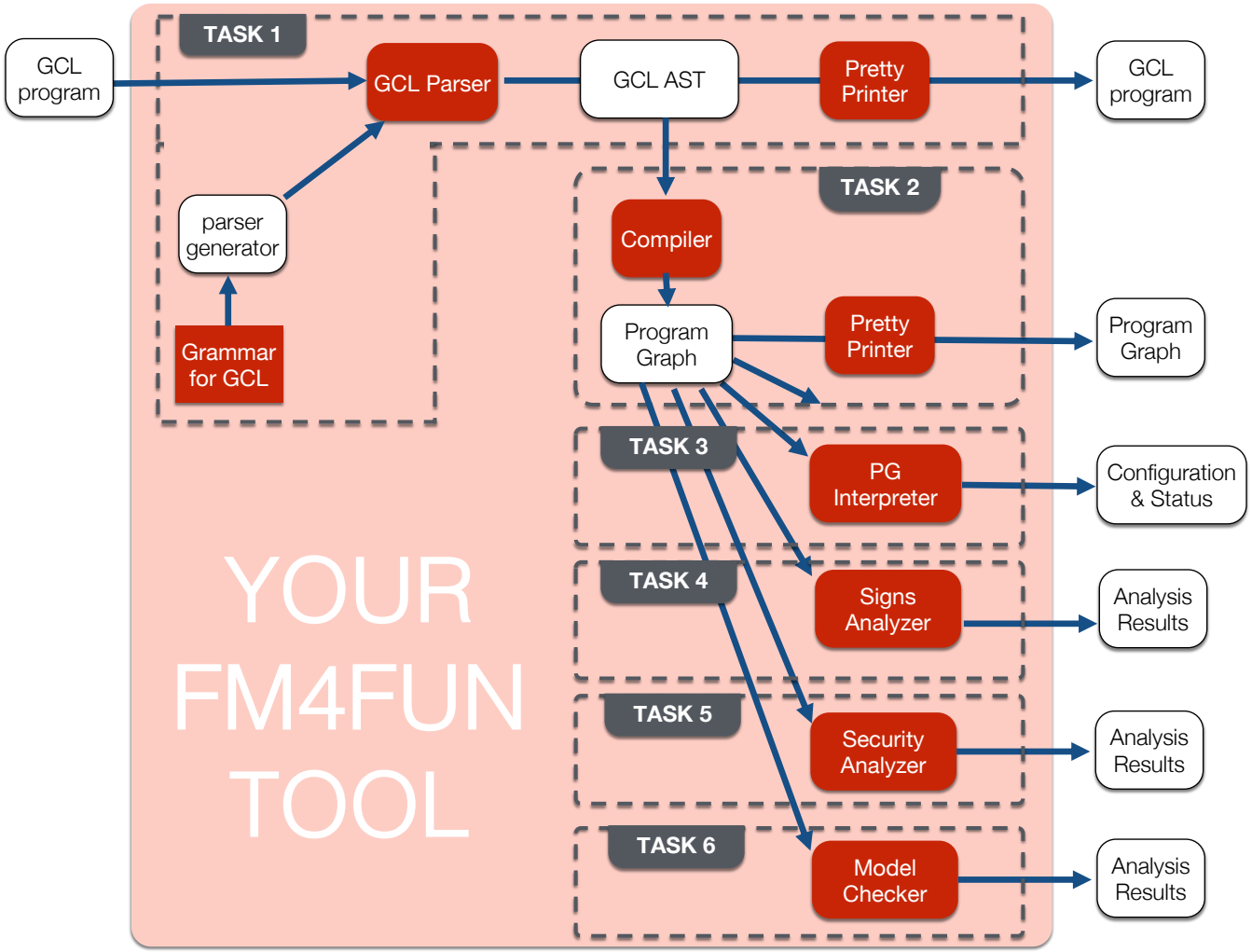
Example: A compiler



- The strings have structure, can be analysed to extract meaning
- Each string is an element of some "language"

→ A strong influence from Linguistics

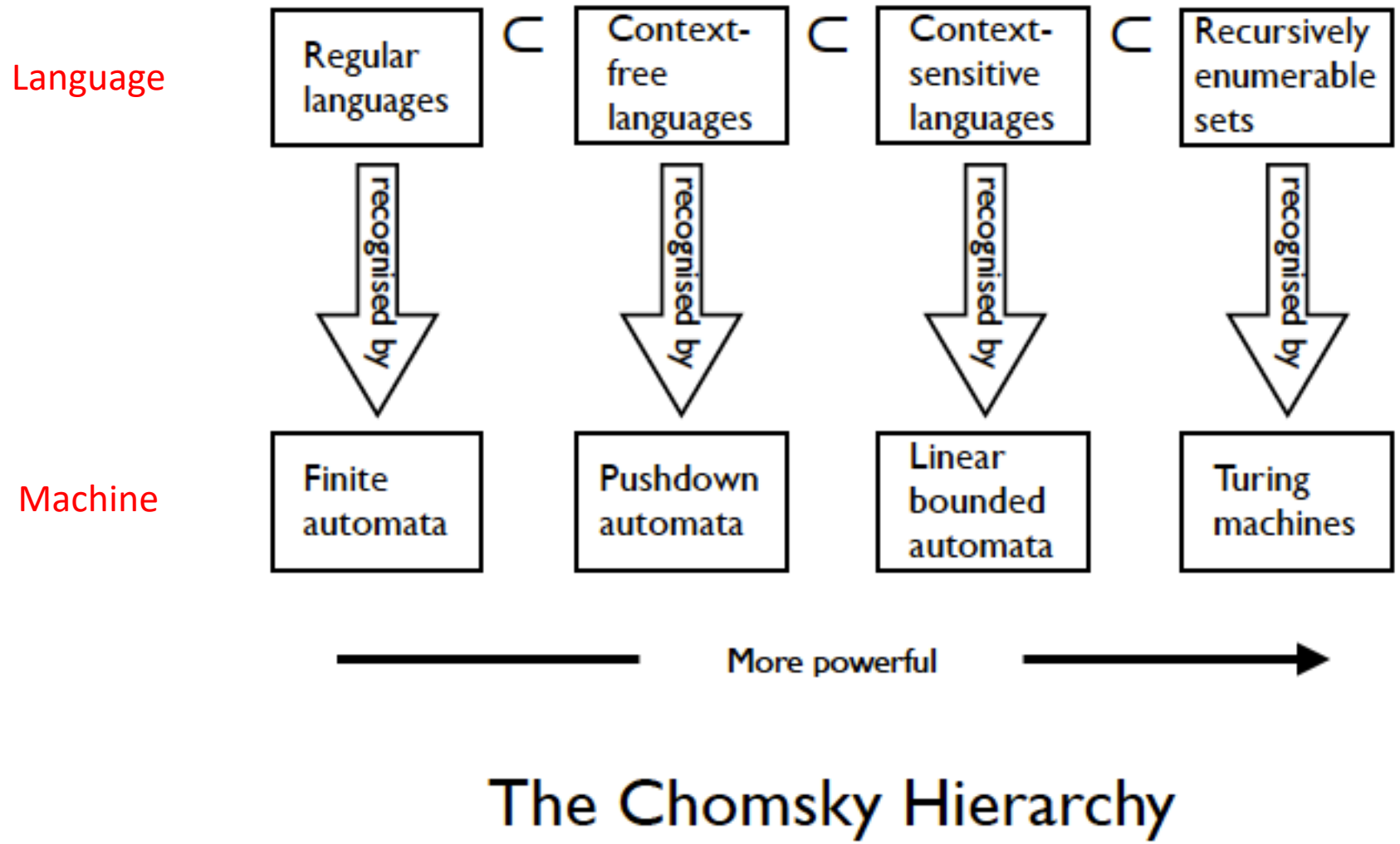
Example: your Mandatory Assignment



Two strands of work pulled together

- **Chomsky**: the languages humans invented for communication with one another and the languages needed for humans to communicate with machines share many basic principles
- **Turing**: if some human can figure out an algorithm for solving a particular class of problems then the machine can be told to follow the steps and execute the algorithm exactly and precisely

Models have different expressive power



The Chomsky Hierarchy

PRACTICAL INFORMATION

The plan for the course

- Regular languages (RL)
  - Lecturer: Alberto Lluch Lafuente
- Context free languages (CFG)
  - Lecturer: Alberto Lluch Lafuente
- Formal Methods (FM)
  - Lecturer: Flemming Nielson
- Mandatory Assignments (MA)
  - Lecturer: Alberto Lluch Lafuente

The plan for the course

DTU Day	Date	M#	Topic	Lecturer	Description	Reading material
Tuesday	4 February 2020	1	RL-1	ALL	Introduction	HMU 1.1 1.5 2.1
Friday	7 February 2020	2	RL-2	ALL	Finite Automata	HMU 1.4 2.2 2.3
Tuesday	11 February 2020	3	RL-3	ALL	Regular Expressions	HMU 3.1 3.2 3.4
Friday	14 February 2020	4	CFL-1	ALL	Introduction to Context Free Grammars	HMU 5.1 5.2
Tuesday	18 February 2020	5	FM-1	FN	Formal Methods, Program Graphs	Quanta Magazine, FM 1.1-1.3
Friday	21 February 2020	6	FM-2	FN	Guarded Commands	FM 2.1-2.3
Tuesday	25 February 2020	7	FM-3	FN	Program Graphs, Guarded Commands	FM 1.4 2.4 (2.5)
Friday	28 February 2020	8	CFL-2	ALL	Grammars and data, ambiguities, associativity and precede	HMU 5.4
Tuesday	3 March 2020	9	CFL-3	ALL	Parser Generators, abstract syntax and demo	HMU 5.3
Friday	6 March 2020	10	MA-1	ALL	Parsing	*
Tuesday	10 March 2020	11	CFL-4	ALL	Push Down Automata	HMU 6
Friday	13 March 2020	12	MA-2	ALL	Compiling	*
Tuesday	17 March 2020	13	FM-4	FN	Program Verification	FM 3.1-3.3
Friday	20 March 2020	14	MA-3	ALL	Interpreters	*
Tuesday	24 March 2020	15	FM-5	FN	Program Analysis	FM 4.1-4.3
Friday	27 March 2020	16	FM-6	FN	Program Analysis	FM 4.3-4.5
Tuesday	31 March 2020	17	RL-4	ALL	Equivalence Results for Regular Languages	HMU 2.3 2.5 3.2
Friday	3 April 2020	18	MA-4	ALL	Program Analysis	*
Tuesday	14 April 2020	19	FM-7	FN	Language Based Security	FM 5.1-5.2
Friday	17 April 2020	20	FM-8	FN	Language Based Security	FM 5.3-5.4
Tuesday	21 April 2020	21	FM-9	FN	Concurrency	FM 8.1-8.2
Friday	24 April 2020	22	MA-5	ALL	Language Based Security	*
Tuesday	28 April 2020	23	FM-10	FN	Model Checking	FM 6.1-6.3
Friday	1 May 2020	24	FM-11	FN	Model Checking	FM 6.4-6.5
Tuesday	5 May 2020	25	MA-6	ALL	Model Checking	*
"Friday"	12 May 2020	26	*	ALL	Wrapping Up	*

Literature and resources

- [HMU] Hopcroft & Motwani & Ullman: *Introduction to Automata Theory, Languages, and Computation*. A compiled edition for 02141 is available from Polyteknisk Boghandel.
- [FM] Nielson & Nielson: *Formal methods – an appertizer.* To appear 2019. A preliminary version is available on DTU Inside.

- See more on the course web-page http://www.imm.dtu.dk/courses/02141

  and on DTU Inside

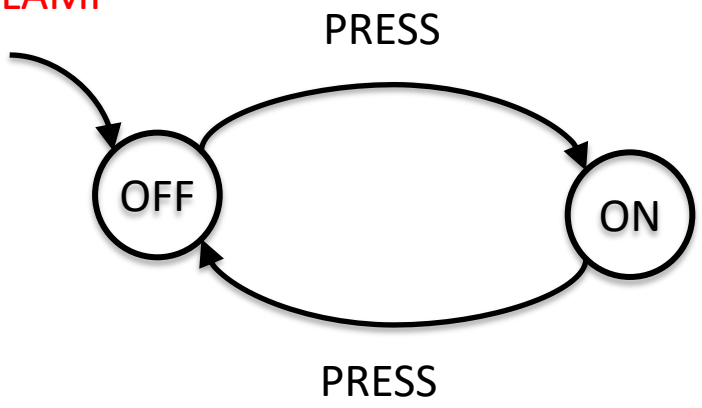
It is strongly recommended to

- Read the material before coming to the lecture – even the parts that you do not fully understand.
- Attend the lectures – look out for explanations of the difficult points and ask questions!
- Form teams of 3 students for the mandatory assignment and for the exercises.
- Reread the material – check that you can explain it to yourself and to your teammates.
- Do the exercises, discuss with your teammates and make sure you get feedback from the teaching assistants and/or from the teacher.

INTRODUCTION TO FINITE AUTOMATA

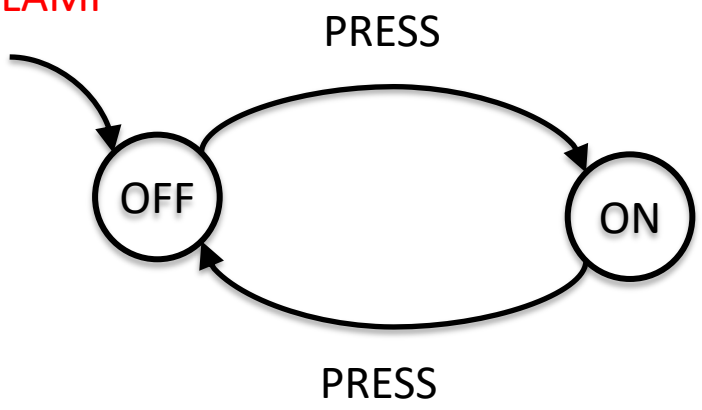
Example: A lamp

BASIC LAMP

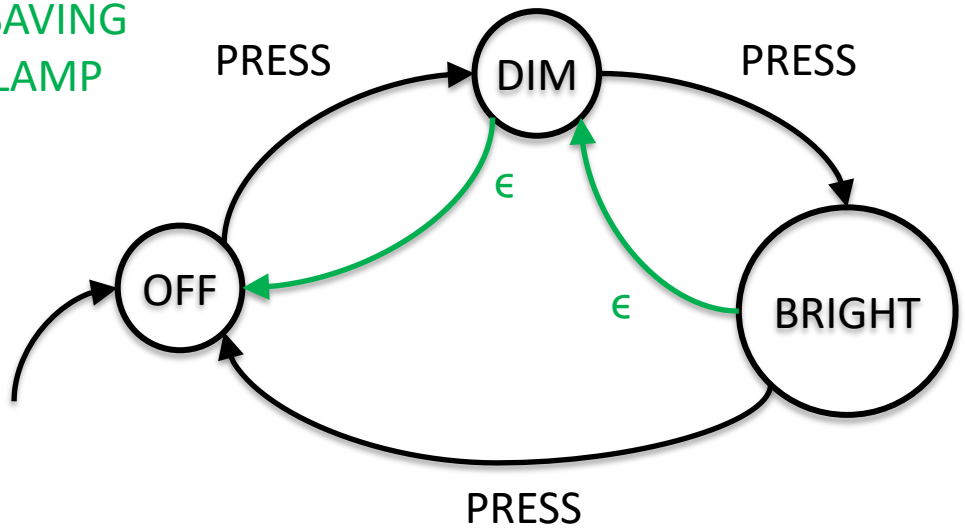


Example: A lamp

BASIC LAMP



ENERGY SAVING SMART LAMP



ϵ are ”internal actions”

Example: A vending machine

- Coffee: 15 kr
- Coins accepted: 5 kr
- Max capacity: 15 kr

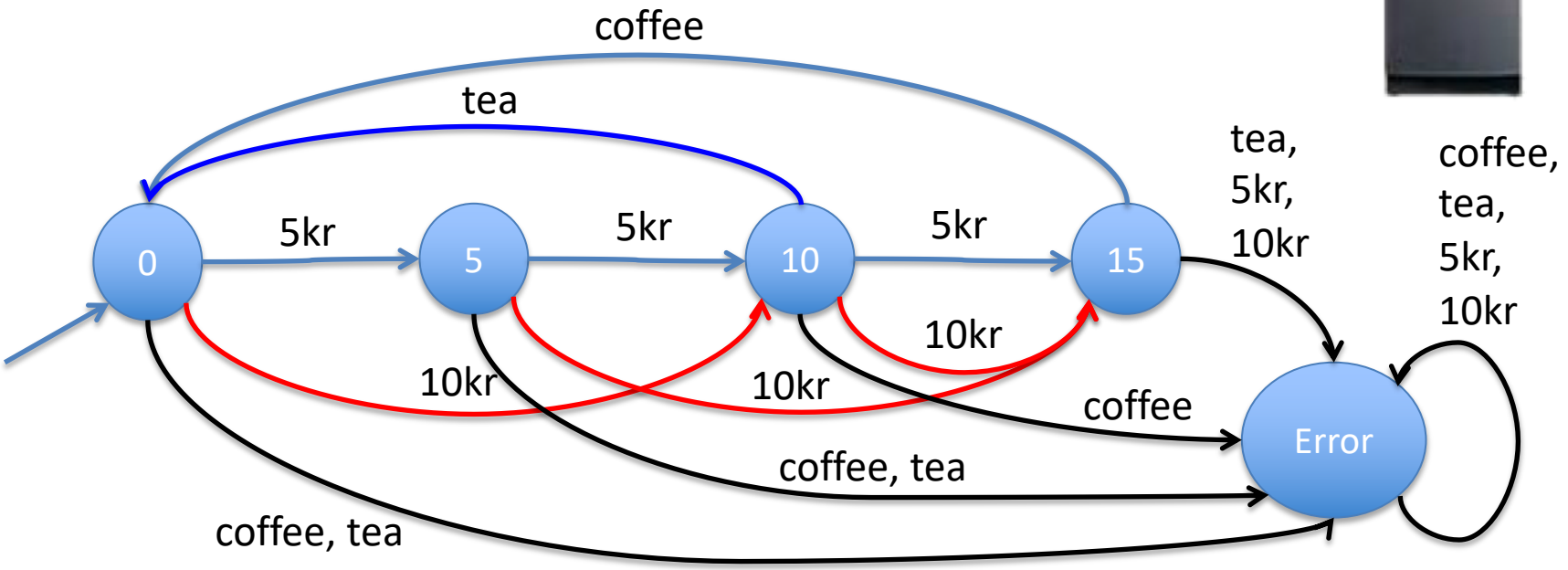
- Also tea: 10 kr
- Also accept: 10 kr

- What happens if we don't follow "the rules"?

- Can we model that we give change back?

Example: A vending machine

A possible model

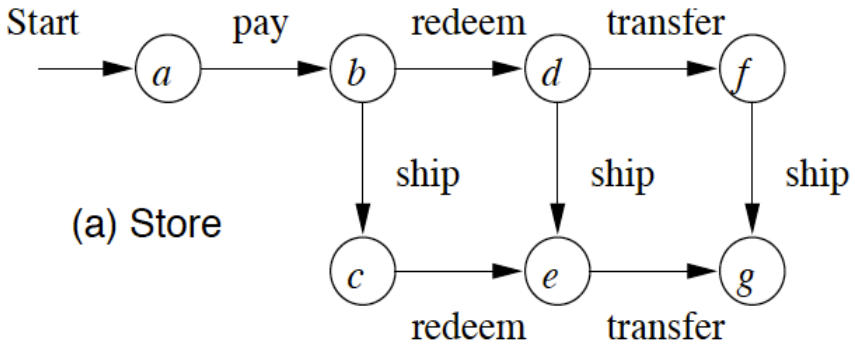


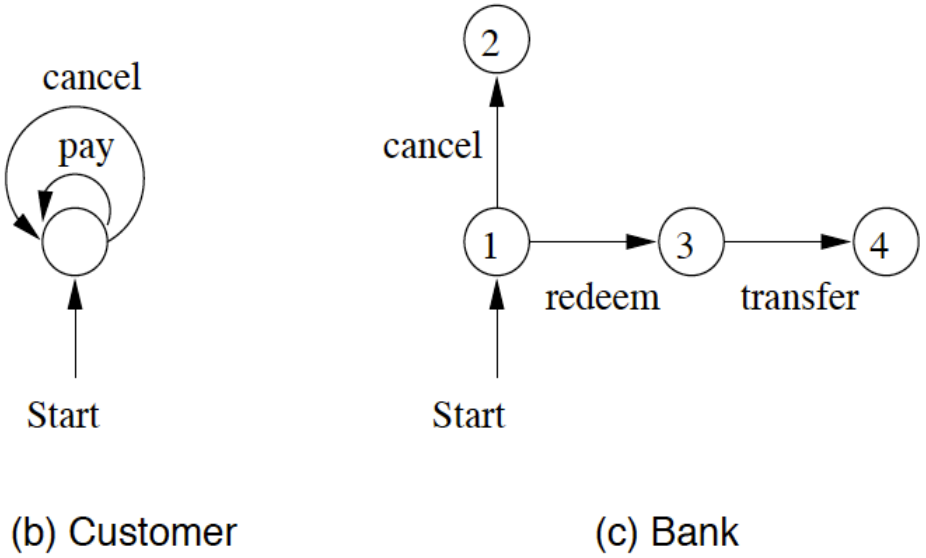
Example from HMU: E-commerce

- Three principals: Customer, Store and Bank
- Model how
  - Customer pays money to Store
  - Customer cancels payment by notifying Bank
  - Store ships goods to Customer
  - Store redeems money from Bank
  - Bank transfers money to Store

Example: E-commerce

Finite automata
for each principal



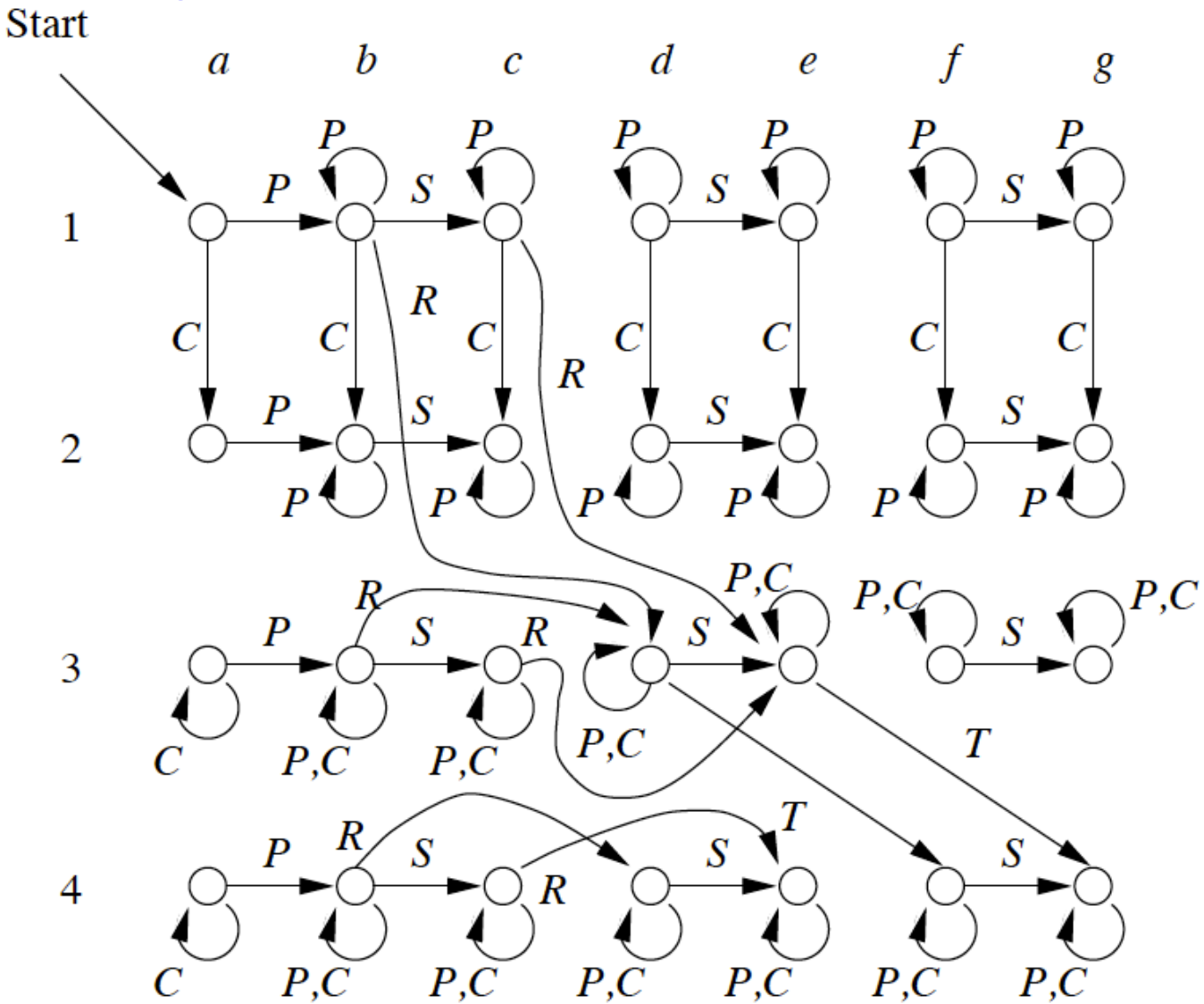


Example: E-commerce

The combined finite automaton

Product automaton: Keep track of the state of all three automata while synchronising on joint actions

P: pay
S: ship
C: cancel
R: redeem
T: transfer



Let's do it!

Exercise: The wolf-goat-cabbage problem

Bring a Ferryman, a Goat, a Cabbage and a Wolf from one side of a river to the other using a small boat that only can carry two of which the Ferryman has to be one.

Left alone the Goat will eat the Cabbage and the Wolf will eat the Goat.

Exercise:

- Construct finite automata for how each of the four individuals may cross the river
- Construct the product automaton
- Identify the dangerous states in the product automaton
- Does there exist a sequence of boat trips that will bring everybody safely across the river?

ALPHABETS, STRINGS AND LANGUAGES

Key concepts

- Σ: alphabet – the actions of the automaton
- Strings are sequences of symbols from an alphabet :
  - w = $a_1\ a_2\ \ldots\ a_k$: a finite sequence of symbols a_i from Σ of length k (≥0)
  - w = ε: the empty sequence (of length 0)
- we write |w| for the length of the string w

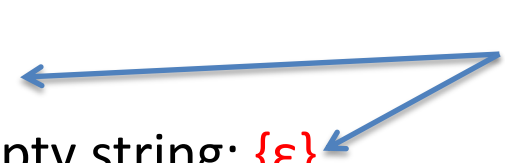
Key concepts

- If x and y are strings, then xy denotes the concatenation of x and y
  - Concatenation is associative $(xy)z = x(yz)$
  - ε is the identity: $\varepsilon w = w\varepsilon = w$
- The lengths add up: $|xy| = |x|+|y|$
- If w is a string we define w^n to be the concatenation of n copies of w:
  - *basis*: $w^0 = \varepsilon$
  - *induction*: $w^{i+1} = w^i w$ for $i \geq 0$

Sets of strings and languages

- Σ^k: the set of strings of length k
- Σ^*: The set of all strings over Σ (with finite length)
  - $\Sigma^* = \Sigma^0 \cup \Sigma^1 \cup \Sigma^2 \cup \ldots$
- Σ^+: The set of non-empty (finite) strings over Σ
  - $\Sigma^+ = \Sigma^1 \cup \Sigma^2 \cup \ldots$

- A language L is a subset of Σ^*
  - Two special cases:
    - The empty language: $\emptyset$
    - The language of the empty string: $\{\varepsilon\}$



Three fundamental problems

- The membership problem:

  Given a language L and a string w, does w belong to L?

- The emptiness problem:

  Is the language L empty?

- The equivalence problem:

  Are the two languages L and M equal?

Three fundamental problems

- The membership problem:

  L1 = all strings that contain at least one occurrence of COFFEE preceded by PRESS.

  L2 = all strings that lead the automaton below to state ☺.

  L3 – all strings that lead the automaton below to state ☺ in just 1 step?

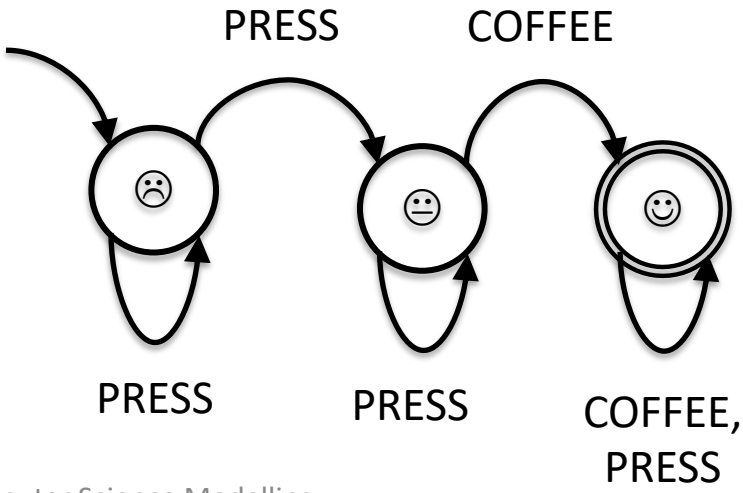
  Does PRESS,PRESS,PRESS,COFFEE belong to L1? And to L2?

- The emptiness problem:

  Is the language L1 empty? And L2? And L3?

- The equivalence problem:

  Are L1 and L2 equal?



Specifying languages

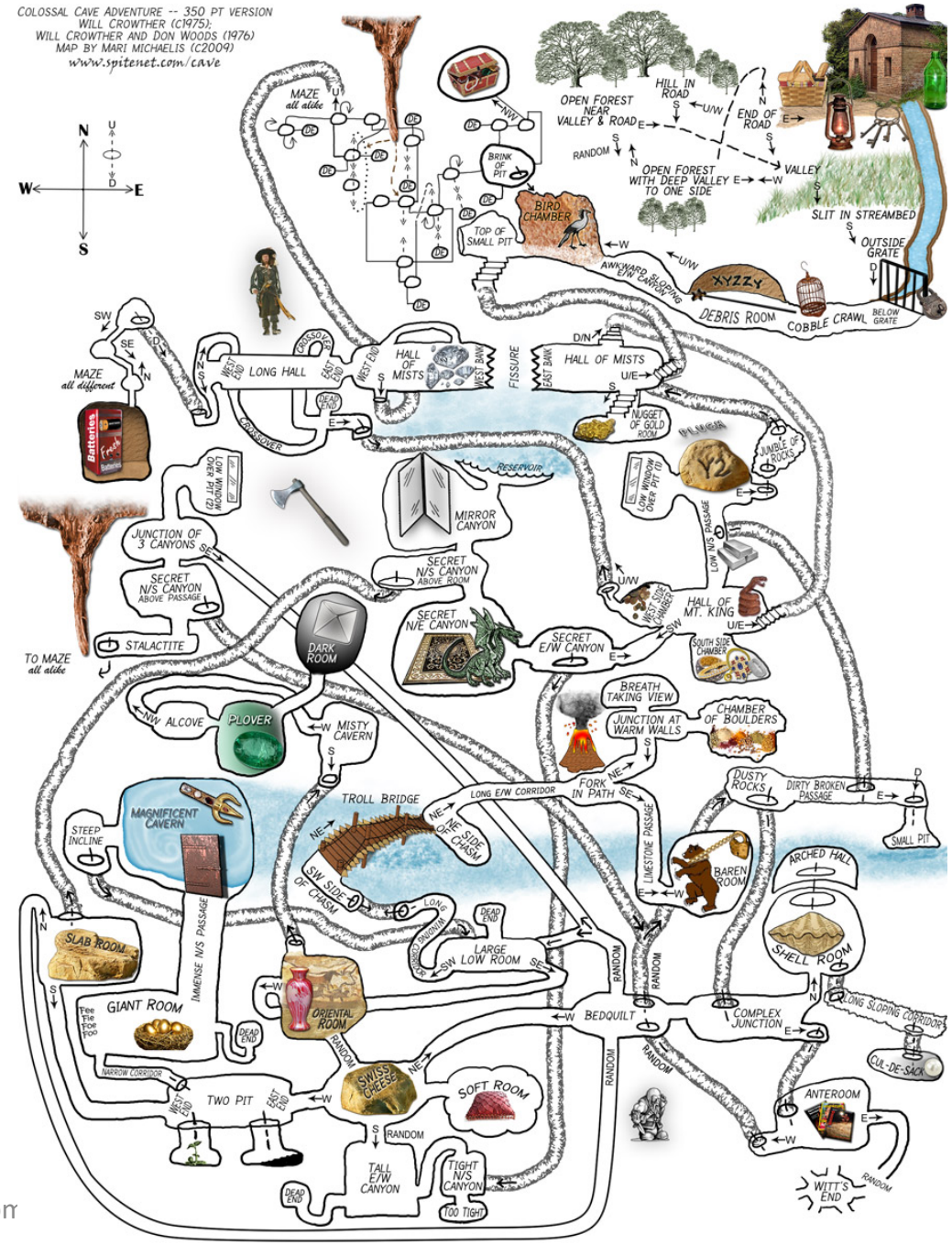
- How is the language specified?
  - Deterministic finite automaton
  - Non-deterministic finite automaton
  - Regular expression

- We shall show that the three methods are equivalent!
- They all specify regular languages

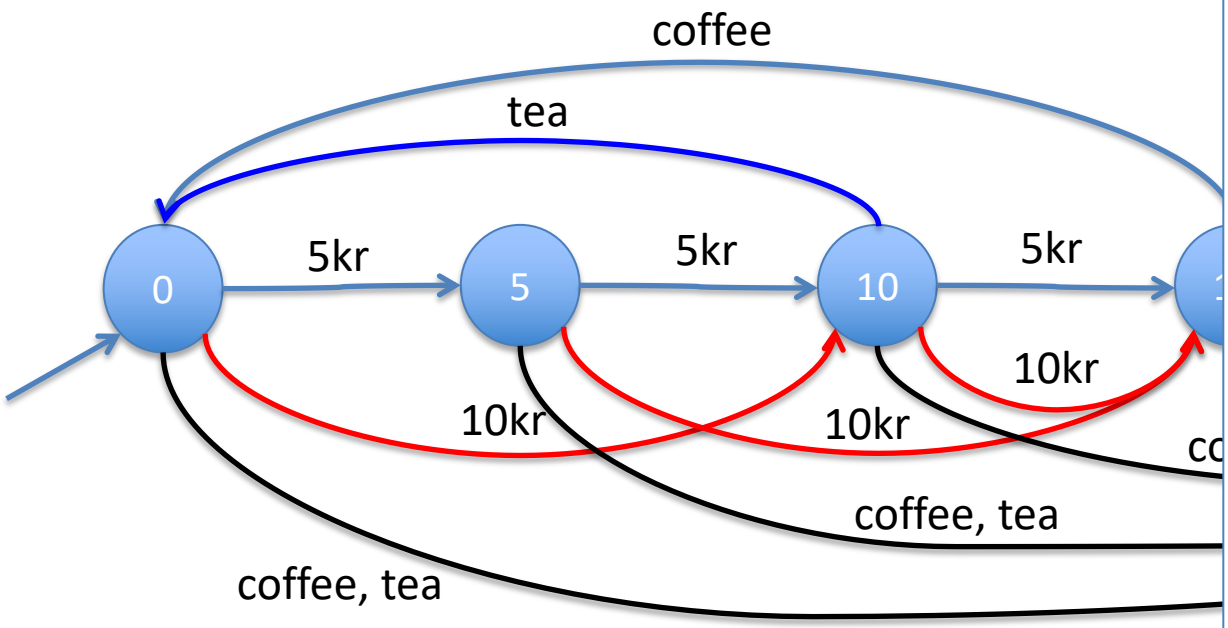
BY THE WAY …
TABLE DRIVEN PROGRAMMING

Colossal Cave Adventure Game

How would you implement the logic of this game?



How would you implement the logic of the vending machine in Java?



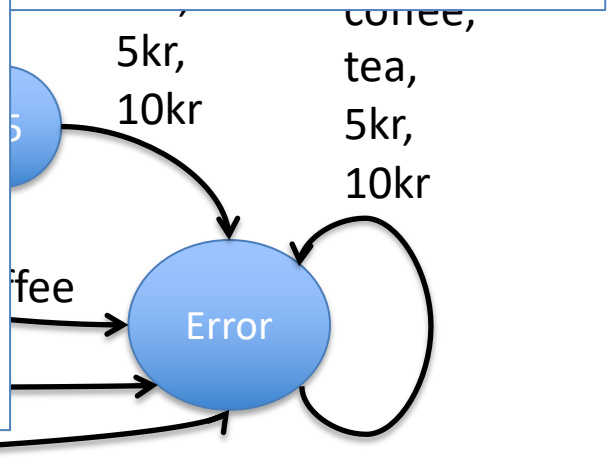

```
public static void main(String [] args){

// Start with the initial state
int S = S0;
System.out.println("Current state is initial state S0");
int I;
int newS = S;

while (true) {
        I = getInput();
        if(S==S0 && I==I5kr) newS = S5;
        else if(S==S0 && I==10kr) newS = S10;
        else if(S==S0 && I==coffee) newS = SERROR;
        else if(S==S0 && I==tea) newS = SERROR;
        else if(S==S5 && I==5kr) newS = S10;
        else if(S==S5 && I==I0kr) newS = S15;
        else if(S==S5 && I==coffee) newS = SERROR;
        else if(S==S5 && I==tea) newS = SERROR;
        else if(S==S10 && I==5kr) newS = S15;
        else if(S==S10 && I==10kr) newS = S15;
        else if(S==S10 && I==coffee) newS = SERROR;
        else if(S==S10 && I==tea) newS = S0;
        else if(S==S15 && I==5kr) newS = SERROR;
        else if(S==S15 && I==10kr) newS = SERROR;
        else if(S==S15 && I==coffee) newS = S0;
        else if(S==S15 && I==tea) newS = SERROR;
        else if(S==SERROR && I==5kr) newS = SERROR;
        else if(S==SERROR && I==10kr) newS = SERROR;
        else if(S==SERROR && I==coffee) newS = SERROR;
        else if(S==SERROR && I==tea) newS = SERROR;

        System.out.println("Transitioned from " +
                           stateNames[S] + " to " +
                           stateNames[newS] +
                           " with input " + inputNames[I]);
        S = newS;
}
```

How would you implement the logic of the vending machine in Java?

```
enum State { S0, S5, S10, S15, ERROR };
enum Input { fivekr, tenkr, coffee, tea };
.
.
.
public static void main(String [] args){

        // Start with the initial state
        State S = State.S0;
        System.out.println("Current state is initial state S0");
        Input I;
        State newS = S;

        while (true) {
          I = getInput();
          switch (S) {
          case S0:
                switch (I) {
                case fivekr:
                  newS = State.S5;
                  break;
                case tenkr:
                  newS = State.S10;
                  break;
                default:
                  newS = State.ERROR;
                  break;
                };
                break;
```

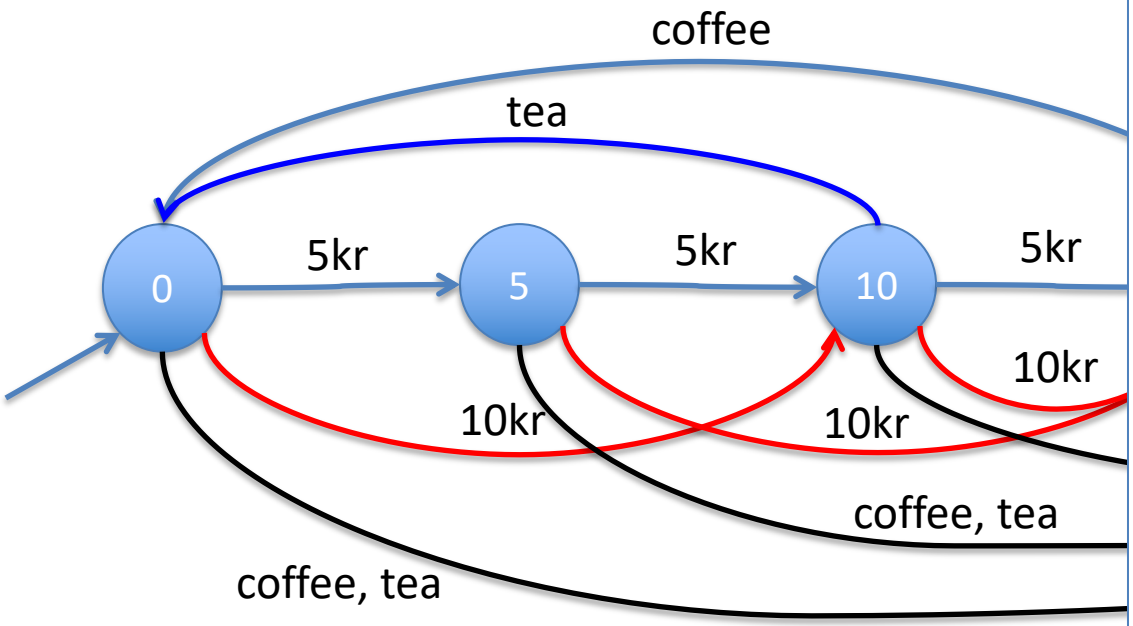
```
case S5:
        switch (I) {
        case fivekr:
          newS = State.S10;
          break;
        case tenkr:
          newS = State.S15;
          break;
        default:
          newS = State.ERROR;
          break;
        };
        break;
case S10:
        switch (I) {
        case fivekr:
        case tenkr:  // 10 + 10 -> 15 (no change)
          newS = State.S15;
          break;
        case coffee:
          newS = State.ERROR;
          break;
        case tea:
          newS = State.S0;
          break;
        };
        break;
```

```
    case S15:
            if (I.equals(Input.coffee))
                    newS = State.S0;
            else
              newS = State.ERROR;
            break;
    case ERROR:
            newS = State.ERROR;
            break;
    };
    System.out.println("Transitioned from " + S + "

                            newS + " with input " + I);
    S = newS;
  }
```



δ	5kr	10kr	coffee	tea
0	5	10	Error	Error
5	10	15	Error	Error
10	15	15	Error	0
15	Error	Error	0	Error
Error	Error	Error	Error	Error

implement the logic of g machine in Java?




```
//The state machine represented by a matrix
//note that the elements on each line below
//represent a column of the matrix
public final static int stateMachine[][] = {
        {S5,S10,S15,SERROR,SERROR},
        {S10,S15,S15,SERROR,SERROR},
        {SERROR,SERROR,SERROR,S0,SERROR},
        {SERROR,SERROR,S0,SERROR,SERROR}};

public static void main(String [] args){

// Start with the initial state
int S = S0;
System.out.println("Current state is initial state 0");
int I;
int newS;

while (true) {
        I = getInput();
        newS = stateMachine[I][S];
        System.out.println("Transitioned from " +
                        stateNames[S] + " to " +
                        stateNames[newS] +
                        " with input " +
                        inputNames[I]);
        S = newS;
}
}
```

READING MATERIAL AND EXERCISES

Reading material and exercises

- Covered in the lecture today:
  - HMU chapter 1, section 1.1 and 1.5
  - HMU chapter 2, section 2.1
  - Topic of the next lecture:

    Finite Automata

    to be based on HMU section 1.4, 2.2 and 2.3
- Exercises for today:
  1. The wolf-goat-cabbage problem (see previous slide)
  2. Implement the vending machine in F#
  3. Extension of the vending machine
     - to offer chocolate at a price of 20kr
     - to accept 20kr coins as well
  4. Model the 2x2 puzzle, the simplest form of the NxN puzzle (https://en.wikipedia.org/wiki/15_puzzle) with an automaton. Which sequences of transitions help you solve the puzzle?