Lecture 7

Methods and problems in Computer Science

Scientific Methods, General

Theoretical methods:

Create formal models (mathematics, logic)

Define concepts within these

Prove properties of the concepts

Abstraction, hide details to make the whole more understandable (and to make it possible to prove properties of it)

Proofs of properties by deductive methods

Empirical methods:

Perform experiments

See how it turned out

Draw conclusions

Simulation:

Start with a formal model at some "easy-to-understand" level

Make "artificial experiments" in your computer

Collect statistics and draw conclusions

In physics:

Make hypotheses about the surrounding world (theory), observe it (experiment)

Relate the result of experiment to theory

Adjust the theory if it doesn't predict the reality well enough

Theory is used to predict the future (e.g., if a bridge will hold for a certain load, or an asteroid fall down on our heads)

Common pattern in Computer Science:

The system is constructed to behave according to some theoretical model

Deviations are seen as construction errors rather than deficiencies in the theory (hardware error, bug in OS, . . .)

In both cases: the theory helps us understand and predict, but in different ways!

Theoretical vs. Empirical Methods in Computer Science

Computer Science really has a “spectrum”, from “extreme constructivism” to a use of theory close the one in physics:

“Extreme constructivism”: (ideal) programming language design:

- Formal semantics for the language, pure construction of model defining the mathematical meaning of each program
- Abstraction of details to make the meaning of the language simpler (for instance, assume that data structures can grow arbitrarily big)
- Implement the language according to the semantics

One can prove formally within the model that a program is correct – valuable!

But the model does not cover all kinds of errors. E.g., hardware errors, or stack overflow (or an asteroid falling down on the computer)

Extreme “physics” approach: performance modelling of complex computer- and communication systems

- Extremely hard to make analytical calculations
- Simplified performance models, tested against experiments (e.g., long suites of benchmarks)
- Discrepancy leads to a modified theory, as in physics
- Often simulation (desire to evaluate systems before building them)

In-between: algorithm analysis

- Build on some form of formal model for how the algorithm executed (metalanguage with formal semantics), and some performance model (how long does a step in the algorithm take, how much memory is needed to store an entity)
- Performance model often of type “one arithmetic operation $=$ one time unit”
- Given that the performance model is correct, one proves mathematically that the algorithm needs certain resources (time, memory) to be carried out
- But the performance model is often very approximate
- Sometimes possible to refine the performance model, but this can make it impossible to calculate the resource needs of the algorithm

Data mining
observational vs. experimental data

Data mining

The process of **automatically** discovering non-**trivial useful** information in **large data repositories.**

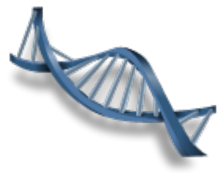
Two aspects of data mining

- **Predictive**
  - Classification
  - Regression
- **Descriptive**
  - Association rules
  - Clustring
  - Anomaly detection
  - Visualisation

Why data mining?

Scientific answer:

- Huge amounts of data are continuously being collected (GB/h)
  - satellite sensors
  - radar telescopes
  - simulation data
  - DNA experiments
- Traditional statistical methods impractical
- Data mining can help scientists to
  - explore, cluster and classify data
  - formulate hypotheses



Why data mining?

Commercial answer:

- Huge amounts of data concerning:
  - purchases
  - surfing och searching the Internet
  - bank and credit card transactions
- Computers have become and more powerful
- Commercial pressure to provide better and customized services

Data mining?

Exercise:

Give an example of something you did today or yesterday that resulted in data that could be mined to discover useful information.

Classification: Finding tax evaders

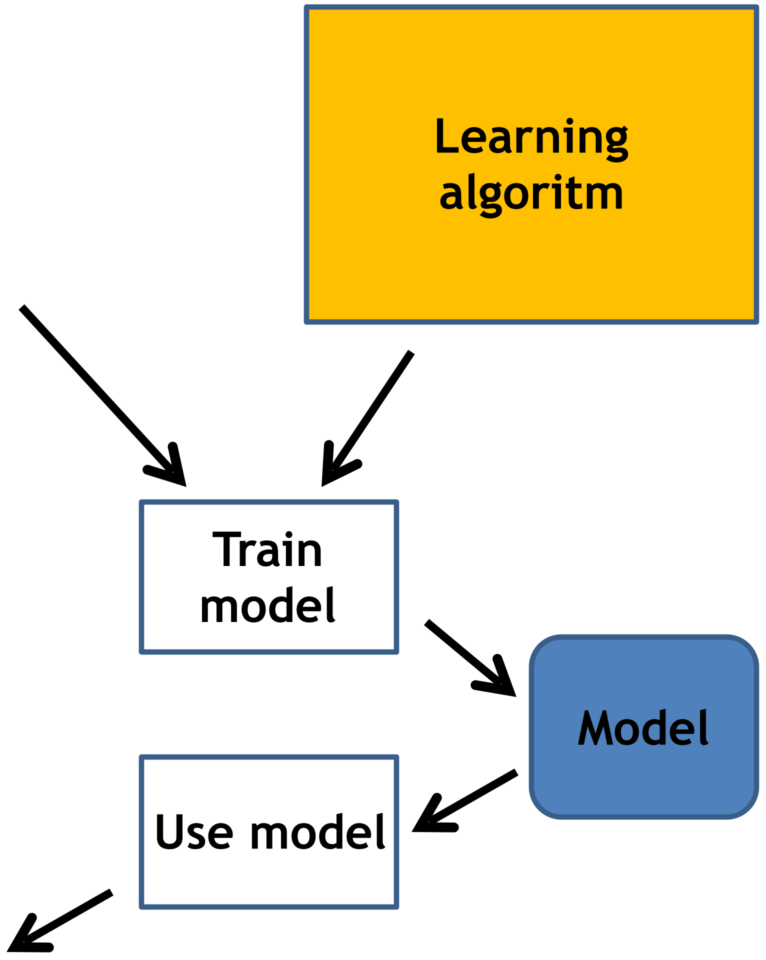
#	Refund	Marital status	Income	Cheated ?
1	Yes	Single	600K	No
2	No	Married	400K	No
3	No	Single	300K	No
4	Yes	Married	420K	No
5	No	Skild	380K	Yes
6	No	Married	220K	No
7	Yes	Skild	800K	No
8	No	Single	360K	Yes
9	No	Married	240K	No
10	No	Single	340K	Yes

Jim: No refund, divorced, earns 120K?

Classification

#	Refund	Marital status	Income	Cheated?
1	Yes	Single	600K	No
2	No	Married	400K	No
3	No	Single	300K	No
4	Yes	Married	420K	No
5	No	Skild	380K	Yes
6	No	Married	220K	No
7	Yes	Skild	800K	No
8	No	Single	260K	Yes
9	No	Married	240K	No
10	No	Single	360K	Yes

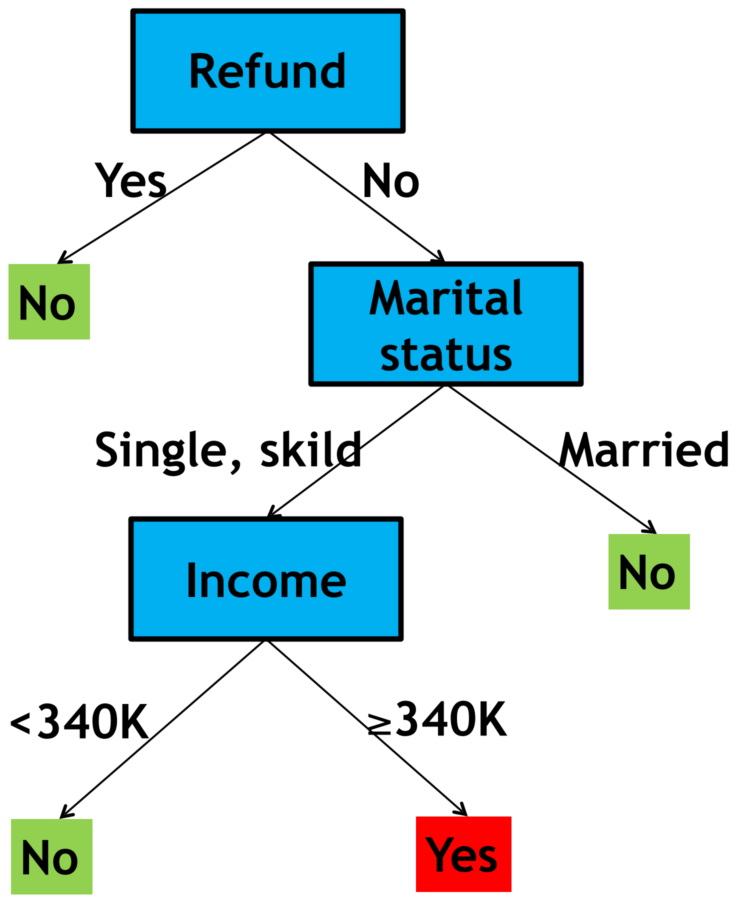
11	No	Married	250K	?



Decision tree

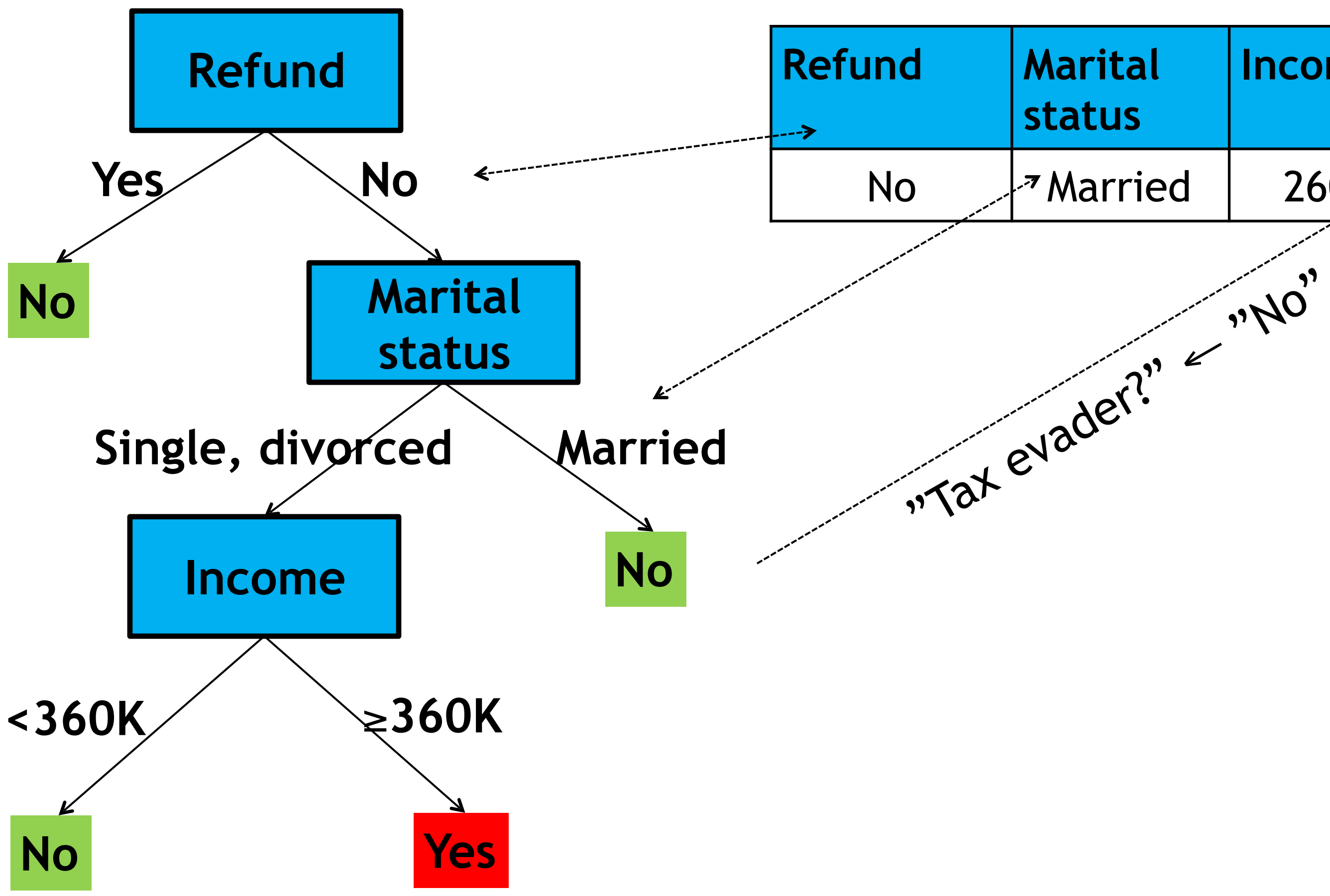
#	Refund	Marital status	Income	Cheated ?
1	Yes	Single	600K	No
2	No	Married	400K	No
3	No	Single	300K	No
4	Yes	Married	420K	No
5	No	Skild	380K	Yes
6	No	Married	220K	No
7	Yes	Skild	800K	No
8	No	Single	360K	Yes
9	No	Married	240K	No
10	No	Single	340K	Yes

Träningsdata



Modell: beslutsträd

Using the tree for classification



Refund	Marital status	Income	Cheated?
No	Married	260K	?

Observational vs. experimental data

- Data mining yields **observational** data
- Observational data can be user to infer **correlations** between variables
- Experimental data can be used to infer causal relationships (cause → effect)

Experiments vs. data mining

Experiments:

- We know what we are looking for
  - Formulate null hypothesis
  - Sampling
  - Reject or accept the hypothesis
- Systematically vary the predictor variables and study the effect on the result variable.

Data mining:

- We do **not** know what we are looking for
- Data come from uncontrolled observations

Observational data

- Which conclusions can be drawn from the following observations:
  - Autopsies show that deceased patients who have suffered from Alzheimer's disease have high levels of aluminium residues in their brains.
  - Historical data show high levels of CO_2 in the atmosphere during periods of increased average temperature.
  - A questionnaire show that obese persons tend to prefer Coke Light before ordinary Coke.
  - A French consumer organisation reported that owners of red cars were more likely to default on their car loans.

Observational data

- Which conclusions can be drawn from the following observations:
  - Autopsies show that deceased patients who have suffered from Alzheimer’s disease have high levels of aluminium residues in their brains.
  - Historical data show high levels of CO_2 in the atmosphere during periods of increased average temperature.
  - A questionnaire show that obese persons tend to prefer Coke Light before ordinary Coke.
  - A French consumer organisation reported that owners of red cars were more likely to default on their car loans.

Observational data

- Which conclusions can be drawn from the following observations:
  - Autopsies show that deceased patients who have suffered from Alzheimer’s disease have high levels of aluminium residues in their brains.
  - Historical data show high levels of CO_2 in the atmosphere during periods of increased average temperature.
  - A questionnaire show that obese persons tend to prefer Coke Light before ordinary Coke.
  - A French consumer organisation reported that owners of red cars were more likely to default on their car loans.

Observational data

- Which conclusions can be drawn from the following observations:
  - Autopsies show that deceased patients who have suffered from Alzheimer’s disease have high levels of aluminium residues in their brains.
  - Historical data show high levels of CO_2 in the atmosphere during periods of increased average temperature.
  - A questionnaire show that obese persons tend to prefer Coke Light before ordinary Coke.
  - A French consumer organisation reported that owners of red cars were more likely to default on their car loans.

Experiment

- Will a daily dosis of vitamin C lead to fewer infections?
- How can we design an experiment to test this?
- **Suggestion 1:** Do a web questionnaire
  - ”Vitamin C makes me healthier.”
  - ”Vitamin C doesn’t affect my health.”
- **Suggestion 2:** Gather some subjects and have them take a daily dosis of vitamin C during a couple of months. Then evaluate whether the subjects have had fewer days of infection compared to the corresponding period the preceding year.

Experiment

- **Suggestion 3:** Gather some subjects. Let each person decide whether she wants to take a daily dosis of vitamin C (group C) or not (group N). At the end of the trial period, we measure whether group C had fewer days of infection than group N.

Experiment

- **Suggestion 4:** Find some subjects. Let the **experiment leader** decide who is going to have a daily dosis of vitamin C (group C) and who will not (group N). At the end of the trial period, we measure whether group C had fewer days of infection than group N.

Experiment

- **Suggestion 5:** Find some subjects . **Randomly** decide who is going to have a daily dosis of vitamin C
  (group C) and who will not (group N). At the end of the trial period, we measure whether group C had fewer days of infection than group N.

Experiment

- **Suggestion 6:** Find some subjects . **Randomly** decide who is going to have a daily dosis of vitamin C
  (group C) and who is going to have a pill that doesn’t contain any active ingredient (group P). The subjects **do not know** whether they belong to group C or group P. At the end of the trial period, we measure whether group C had fewer days of infection than group P.

Experiment

- **Suggestion 7:** Find some subjects . **Randomly** decide who is going to have a daily dosis of vitamin C
  (group C) and who is going to have a pill that doesn’t contain any active ingredient (grupp P). The subjects **do not know** whether they belong to group C or group P, and **neither does the experiment leader.**
  At the end of the trial period, we measure whether group C had fewer days of infection than group P.

Design principles for experiments

- Control group
- Randomly select who is part of the experiment group and who is part of the control group
- Placebo
- Double blind tests

Theoretical Models in Computer Science

Discrete mathematics: basic set theory, relations, functions, graphs, algebra, combinatorics, category theory, etc.

The *science* logic: different logical systems, how to make “proofs about proofs”

Theory for complete partial orders (formal semantics)

Topology (mathematics with notions of distance and convergence)

Probability theory, statistics

(Traditional analysis)

Theoretical ***Problems*** in Computer Science

What do we want to prove theoretically within Computer Science?

For instance properties of programs, systems, algorithms, and problems

Some examples:

“FFT uses $O(n \log n)$ operations”

“With 99% confidence the program p runs faster than 1.3 ms on machine m”

“The program p terminates for all indata”

“If the method M says that a program terminates then this is true”

“There is no method that can decide, for any program, whether it terminates or not”

“For each CREW PRAM-algorithm there is an EREW PRAM-algorithm that can simulate it with a certain slowdown”

$P = NP$ (or $P \neq NP$)

“The two semantics S_1 och S_2 agree for each program in the programming language P”

Deductive Methods in Computer Science

1. “Ordinary” mathematical proofs:

- Often *finite entities*: defined *recursively*, properties proved with *induction*
- But also reasoning about limits (“go to the limit”), when infinite behaviours are modelled
- Encodings and translations – common in Complexity Theory
- Sometimes also more conventional mathematical techniques

2. Direct modelling with logical inference systems:

- Common in semantics of programming languages (operational semantics)
- Proof methods from logic (proofs about proofs), again induction!

Let's see some examples. . .

Example [illegible] Algorithm Analysis

Purpose: to find the *cost* of executing an algorithm (that solves a given problem)

(Archetypal problem: to sort a sequence of numbers)

Cost is typically *running time*, but can also be memory requirements, power consumption, etc.

To calculate the cost requires:

- a *machine model*
- a *notation*, i.e., “programming language” for the machine

Typically only interested in the *asymptotic complexity* of the algorithm

“How fast does the execution time grow with the size of the input?”

An example: insertion sort

```
for j = 2 to length(A) do
  key = A[j]
  i = j - 1
  while i > 0 and A[i] > key do
    A[i+1] = a[i]
    i = i - 1
  A[i+1] = key
```

We want to find the execution time as a function of input size (`length(A)`)

Let us informally analyze insertion sort!

Assume execution time of a program is sum of the time of all executions of individual statements,

and that the execution time of an individual statement is constant

(How reasonable is this assumption, really?)

Thus, we can, for each statement take its execution time times the number of times it is executed, and then sum over all statements

Say statements 1 - 7 have execution times $c_1, \ldots, c_7$

Each statement s is executed t_s times

Then total execution time is

$$\sum_{s=1}^{7} t_s \cdot c_s$$

Let's calculate the different t_s on wyteboard and see what we get. . .

Results of analysis:

Best-case execution time (with $n =$ length of A):

$c_1 n + (c_2 + c_3 + c_4 + c_7)(n-1)$

order $\Theta(n)$ (what do we mean by this?)

Worst-case execution time:

$(\frac{c_4}{2} + \frac{c_5}{2} + \frac{c_6}{2})n^2 + (c_1 + c_2 + c_3 + \frac{c_4}{2} - \frac{c_5}{2} - \frac{c_6}{2} + c_7)n - (c_2 + c_3 + c_4 + c_7)$

order $\Theta(n^2)$

Average-case execution time:

order $\Theta(n^2)$

What kind of mathematics did we use?

Proving $h \in \Theta(f)$ is done by ordinary mathematical methods (reasoning about inequalities, deciding the existence of certain entities, ...)

Facts about sums, algebraic manipulations

Probability theory to get the average-case execution time

In short, traditional mathematics

Note, though, that certain details are swept under the carpet!

In particular, implicit assumptions about semantics of loops etc. (how do we know the body of `for j = 2 to n` is executed exactly $\mathtt{n} - 1$ times?)

Example 3: Complexity Theory

Deals with *problems*, or *classes* of problems, rather than single algorithms

Tries to find limits for how costly a certain problem (or class of problems) is on a certain machine model

An example of a problem is sorting:

- $O(n \log n)$ algorithms are known (for sequential machine model)
- Not proved whether this is the ultimate lower limit!

A famous class of problems (a *complexity class*):

NP, the set of all problems that can be solved in polynomial time ($O(n^k)$ for some k) by a non-deterministic Turing machine ($\approx$ set of problems solvable by “brute force parallel search” in polynomial time)

A “hardest” problem in NP is known, 3-SAT: if 3-SAT always can be solved in polynomial time then *each* problem in NP can be solved in polynomial time

Proof by encoding: that each problem in NP can be translated into 3-SAT such that a solution of the translated problem solves the original problem (in polynomial time relative to the time to solve the translated problem in 3-SAT)

(Or the reverse: if there is *any* problem in NP that *cannot* be solved in polynomial time, then 3-SAT cannot either!)

3-SAT is NP*-complete*

Proof that *another* problem Q is NP-complete:

1. Show $Q \in NP$
2. Show that if one can solve Q in (sequential) polynomial time then 3-SAT can be solved in polynomial time (via translation of 3-SAT into Q)

Complexity theory uses *encodings* a lot

Another famous complexity class:

P, the set of all problems that can be solved in polynomial time by a *deterministic* Turing machine (cf. NP)

The class of problems that can be solved *sequentially* in polynomial time (like, for instance, sorting)

Open question: is $P = NP$?

Generally assumed that $P \neq NP$, but has not been proved!

If indeed $P = NP$, then the concept of NP-completeness becomes quite meaningless

Recursive Definitions and Proofs by Induction

Induction over natural numbers

Show that the property P is true for all natural numbers (whole numbers ≥ 0)

1. Show that P holds for 0
2. Show, for all natural numbers n, that if P holds for n then P holds also for $n+1$
3. Conclude that P holds for all n

Formulated in formal logic:

$$[P(0) \land \forall n.P(n) \implies P(n+1)] \implies \forall n.P(n)$$

Example: show that for all natural numbers n holds that

$$\sum_{i=0}^{n}(2i-1) = n^2 - 1$$

Why does induction over the natural numbers work?

The set of natural numbers $\mathbf{N}$ is an *inductively defined set*

(A variation of) Peano's axiom:

- $0 \in \mathbf{N}$
- $\forall x . x \in \mathbf{N} \implies s(x) \in \mathbf{N}$
- $\forall x . 0 \neq s(x)$
- $\forall x, y . x \neq y \implies s(x) \neq s(y)$

$s(x)$ "successor" to x, or $x + 1$

$$\begin{array}{ccccccccc} 0 & \rightarrow & s(0) & \rightarrow & s(s(0)) & \rightarrow & s(s(s(0))) & \rightarrow & \cdots \\ 0 & & 1 & & 2 & & 3 & & \cdots \end{array}$$

Note how proofs by induction over the natural numbers follow the structure of their definition

$$\begin{array}{ccccccccc}
0 & \rightarrow & s(0) & \rightarrow & s(s(0)) & \rightarrow & s(s(s(0))) & \rightarrow & \cdots \\
0 & & 1 & & 2 & & 3 & & \cdots \\
P(0) & \Longrightarrow & P(1) & \Longrightarrow & P(2) & \Longrightarrow & P(3) & & \cdots
\end{array}$$

Also note that the definition of $\mathbf{N}$ is given a well-defined meaning by Kleene's fixed-point theorem:

$\emptyset \subseteq \{0\} \subseteq \{0,1\} \subseteq \{0,1,2\} \subseteq \cdots$

$List$ is a kind of *abstract data type* – the internal representation is hidden

Need *not* be represented as linked structures in memory (but could be)

Typical elements in $List$: NIL, $3 : (4 : NIL)$

Note similarity with the set of natural numbers

$List$ is the set of finite (but arbitrarily long) lists of numbers

Inductively defined sets are typically sets of *infinitely* many *finite* objects

Entities in Computer Science are often finite (data structures, programs, ...)

Example: mathematical definition of the set of (finite) lists of integers, $List$

- $NIL \in List$
- $z \in \mathbf{Z} \wedge l \in List \implies z : l \in List$
- $\forall z, l.(z : l \neq NIL)$
- $\forall z, z', l, l'.(z : l = z' : l' \implies z = z' \wedge l = l')$

We can define mathematical functions over lists. An example:

$$\begin{aligned} length(NIL) &= 0 \\ length(z:l) &= 1 + length(l), \quad \text{for all } z \in \mathbf{Z} \text{ och } l \in List \end{aligned}$$

Defines $length$ as a function $List \to \mathbf{N}$

Recursive definition: $length$ itself is used in the definition! (Seemingly circular definition, but note that $length$ is not applied on the same argument in the right-hand side)

Exercise: show that $length$ really is a well-defined partial function! (That is, that each function value is uniquely determined by the definition.)

Note the similarity with function definitions in some functional languages

Exercise: show $\forall l.length(l) \geq 0$

How to do this?

Each inductively defined set has an *induction principle* that follows the inductive definition of the set. Induction is performed on the “pieces” of an entity built up from smaller entitites (e.g., a list built of elements put in front of shorter lists).

Induction principle for $List$. Show that the property P is true for all lists of integers:

1. Show that P holds for NIL
2. Show, for all lists l and integers z, that if P holds for l, then P holds also for $z : l$
3. Conclude that P holds for all lists of integers

"Mathematical" lists, and functions like $length$, can be seen as *abstract specifications* of what lists are and how functions on them should work

Consider the following piece of C code:

```
#define NIL 0

struct list
{ int contents;
  list *succ;
}

int len(list *l)
{ int length;
  length = 0;
  while(l != NIL)
    {length++; l = l -> l.succ;}
  return(length);
}
```

Interesting things to verify:

- That lists of `list`-structs represent “mathematical” lists in $List$ correctly
- That `len(l)` $= length(l)$ always, when `l` is the representation in C of l

The verification requires that a *formal semantics* is defined for C programs, and that we define exactly what it means that a C entity represents a “mathematical” entity

Logic deals with *formal systems for derivations*, that is, “how to prove things”, and *properties of derivations* (proofs).

Thus, logic is a *metatheory*, which deals with properties of other theories!

Example of a result in logic: “in all logical systems that can express arithmetics on whole numbers, it is possible to formulate statements that can neither be proved nor disproved” (Gödel’s incompleteness theorem)