A refresher on induction

The induction principle is very useful, as you all probably know. Let's refresh it.

- Proof method

To show that a property, say P, holds of every natural number n (i.e., to prove P(n) for all n) it suffices to show that

- P(0) is true &
- for all k, P(k) implies P(n+1)

Example: for all n, sum(n) = n(n+1)/2 where sum(k) = 1 + ... + k

- sum(0) = 0 = 0(0+1)/2
- for all k, if sum(k) = k(k+1)/2 then

sum(k+1) = sum(k) + (k+1)	by definition
= k(k+1)/2 + (k+1)	by inductive hypothesis
= (k(k+1) + 2(k + 1)) / 2	by arithmetic laws
= (k + 1)(k+2)/2	by distributivity of multiplication over sum on natural numbers

- Definitional mechanism

To define a set S inductively using a finite number of constructors f1,....,fn each with a finite arity on a set of 'basic elements'

- fix a set I of basic elements (you can think of the elements in I as constructors of arity 0)	basis
- if e1,...,ek are in S and f is a constructor of arity k then f(e1,...,ek) is an element of S	induction
- nothing else can be an element of S	closure

- Example: I ={0} and s(_) is a constructor of arity 1, then the inductively defined set S = {0, s(0), s(s(0)), ...} is isomorphic to natural numbers
(Indeed basis / induction / and closure boil down to the axioms of Peano).

An exercise in axiomatic semantics

m1: map f [] = []
m2: map f a:as = f(a):(map f as)

Example: double x = x+x => map double [1,2,3] = [2,4,6]

i1: inverse [] = []
i2: inverse a:as = (inverse as) ++ [a]

Example: inverse [1,2,3] = [3,2,1]

Exercise 1
Give an inductive definition of the set of lists of natural numbers.

Prove that for all functions f and all lists as, inverse (map f as) = map f (inverse as)

inverse (map f []) = inverse []	by m1	map f (inverse []) = map f []	by i1
= []	by i1	= []	by m1

inverse (map f a:as) = inverse (f(a):(map f as))	by m2
= (inverse (map f as)) ++ [f(a)])	by i2
= (map f (inverse as)) ++ [f(a)]	by inductive hypothesis
= map f ((inverse as) ++ [a])	by lemma1: (map f as) ++ (map f bs) = map f (as ++ bs)
= map f (inverse as) ++ (inverse [a]))	by lemma2: if len(as) = 1 then inverse as = as
= map f (inverse a: as)	by lemma3: (inverse as) ++ (inverse bs) = inverse (bs ++ as)

Exercise 2
Prove lemmas 1, 2, and 3 above.

What do we mean by correctness?

Safety: "nothing bad happens"

Examples:

- if a number is printed than it is a prime lower than 10^10
- deadlock freedom

Liveness: "something good happens"

Examples:

- All robots looking for a recharge eventually find a charge station
- if a thread tries to get a number to check for primality, it will get one

By the way:

sequential programs can be thought of as multi-threaded programs made of a single thread

BUT

- testing is hard with concurrency because of heisenbugs
    - poor reproducibility
    - failed tests hardly help bug localisation
- non-determinism is both a blessing and a curse

Modelling behaviour

at some level of abstraction

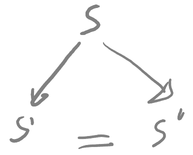
The evolution of a system can be described in terms of state transitions
- states represent the possible configurations the system can be in
- transitions represent the possible evolution from a given configuration.

In its simplest form, such models can be mathematically rendered as binary relations

$Sys = (S, \to)$ where
- S is a set of states ← a.k.a. configurations
- $\to \subseteq S \times S$

$(s, s') \in \to$, usually written $s \to s'$, reads "from state s, Sys can evolve to s'"

Sys is deterministic if $\forall s, s', s'' \in S : s \to s',\ s \to s'' \implies s' = s''$



Of course this idea is hardly new and examples can be found in any book on automata or formal languages. Its application to the definition of programming languages can be found in the work of Landin and the Vienna Group [Lan,Oll,Weg].

[Lan] Landin, P.J. (1966) A Lambda-calculus Approach, Advances in Programming and Non-numerical Computation, ed. L. Fox, Chapter 5, pp. 97–154, Pergamon Press.

[Weg] Wegner, P. (1972) The Vienna Definition Language, ACM Computing Surveys 4(1):5–63.

[Oll] Ollengren, A. (1976) Definition of Programming Languages by Interpreting Automata, Academic Press.

Examples (see [Plotkin])

Finite Automata

$M = (Q, \Sigma, \delta, q_0, F)$

- Q, Σ finite sets
- $q_0 \in Q$, $F \subseteq Q$
- $\delta \subseteq (Q \times \Sigma) \times Q$ or $\delta : Q \times \Sigma \to 2^Q$

$\overset{\text{a corresponding TS}}{\rightsquigarrow}$

$Sys_M = (Q \times \Sigma^*, \to_M)$ where (infinite)

$\to_M = \{((q, aw), (q', w)) \mid q' \in \delta(q, a)\}$

Counters

Operations (infinite)

$\left.\begin{array}{l} \underline{inc}\ i : m \\ \underline{dec}\ i : m \\ \underline{zero}\ i : m / m' \\ \underline{stop} \end{array}\right\} 1 \le i \le n$

Programs = operations*

$\overset{\text{a corresponding TS}}{\underset{\text{fixed } P \in \text{Programs}}{\rightsquigarrow}}$

$Sys_P = (S, \to_P)$ where

- $S = \text{lines}(P) \times \mathbb{N}^n$

  $\{(m, v_1, \dots, v_n) \mid m \in \text{lines}(P)\ \&\ v_i \in \mathbb{N}\quad 1 \le i \le n\}$

- $(m, v_1, \dots, v_n) \to_P (m', v_1', \dots, v_n') \iff$

  $P[m] = \underline{inc}\ i : m'\ \&\ v_i' = v_i + 1\ \&\ \forall j \neq i\ v_j' = v_j$

  or $P[m] = \underline{dec}\ i : m'\ \&\ v_i = v_i' + 1\ \&\ \forall j \neq i\ v_j' = v_j$ (it can't be $v_i = 0$)

  or $P[m] = \underline{zero}\ i : m' / m''$?

  or $P[m] = \underline{stop}$?

> Exercise 3
> Complete the definition of the transition relation $\to_P$ so that such relation is deterministic for all programs