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]

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

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

map f (inverse []) = map f []       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 1: Give an inductive definition of the set of lists of natural numbers and prove lemmas 1, 2, and 3 above.

What do we mean by correctness?

- SAFETY: "nothing bad happens"
  - If a number is printed then it is a positive prime less than 10^{10}
  - No deadlock
- LIVENESS: "something good eventually happens"
  - All robots looking for a recharge eventually find a charge station

BTW: One can think of sequential programs as multi-threaded ones with 1 thread only

But there're serious differences:

- testing is hard { poor reproducibility; failed tests hardly help bug localisation — heisenbugs
- non-determinism: blessing & 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

$Sys = (S, \rightarrow)$ where
- S is a set of states

aka configurations

- $\rightarrow \subseteq S \times S$

$(s, s') \in \rightarrow$ "from the state s, Sys can evolve to s'"

Sys is deterministic if $\forall s, s', s'' \in S : \begin{array}{l} s \rightarrow s' \\ s \rightarrow s'' \end{array} \Rightarrow s' = s''$

$$\begin{array}{ccc} & s & \\ \swarrow & & \searrow \\ s' & = & s' \end{array}$$

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.
[Oll] Ollengren, A. (1976) Definition of Programming Languages by Interpreting Automata, Academic Press.
[Weg] Wegner, P. (1972) The Vienna Definition Language, ACM Computing Surveys 4(1):5–63.

Examples (Plotkin)

Finite Automata (finite)

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

Q, Σ finite $\quad q_0 \in Q \quad F \subseteq Q$

$\delta \subseteq (Q \times \Sigma) \times Q \qquad \delta : Q \times \Sigma \to 2^Q$

a corresponding TS is

- $Q \times \Sigma^*$ (infinite)
- $(q, w) \longrightarrow (q', w') \Longleftrightarrow$

$\longrightarrow = \{((q, aw), (q', w)) \in (Q \times \Sigma^*)^2 \mid q' \in \delta(q, a)\}$

Counters (infinite)

Operations $\quad$ $\underline{inc}\ i : m \qquad 1 \leq i \leq n$

$\underline{dec}\ i : m$

$\underline{zero}\ i : m/m'$

stop

Programs = operations*

(fixed program P)

the corresponding TS is

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

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

- $(m, v_1, \dots, v_n) \longrightarrow_P (m', v_1', \dots, v_n') \Longleftrightarrow$

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

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

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

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

it can't be $v_i = 0$

Exercise 2
Complete the definition of the transition relation $\longrightarrow_P$ so that such relation is deterministic for all programs

Context Free Grammars (infinite)

$G = (N, \Sigma, P, s \in N)$

$P \subseteq N \times (N \cup \Sigma)^*$ s start symbol

(fixed grammar G)
the corresponding TS is

$S = (N \cup \Sigma)^*$

$$\rightarrow_G = \bigcup_{(X,w) \in P} \{ (uXv, uwv) \mid u, v \in (N \cup \Sigma)^* \}$$

Variants

terminal TS $\quad Sys = (S, \rightarrow, T)$ where

$(S, \rightarrow)$ is a TS &

$T \subseteq S$ s.t. $\forall s \in T \; \forall s' \in S \;\; s \not\rightarrow s'$

Exercise 3
Give a terminal TS for each of the examples above