Regular expressions

BNF-like syntax A, finite alphabet

$$E ::= 0 \mid 1 \mid a \mid E+E \mid E\cdot E \mid E^*$$

$a \in A$

end skip atomic instruct. if-then-else -;- iteration

We'll see that this grammar is a Term-Algebra

Denotational semantics: $\mathcal{L} : E \longrightarrow 2^{A^*}$

Term-Algebra homomorphism

$\mathcal{L}(0) = \emptyset$ $\mathcal{L}(1) = \{\varepsilon\}$ $\mathcal{L}(a) = \{a\}$

$$\mathcal{L}(E_1 + E_2) = \mathcal{L}(E_1) \cup \mathcal{L}(E_2)$$

$$\mathcal{L}(E_1 \cdot E_2) = \mathcal{L}(E_1) \cdot \mathcal{L}(E_2) \triangleq \{ vw \in A^* \mid v \in \mathcal{L}(E_1),\ w \in \mathcal{L}(E_2) \}$$

$$\mathcal{L}(E^*) = \mathcal{L}(E)^* = \bigcup_{n \geq 0} \mathcal{L}(E)^n .$$

Exercise 6
Prove or disprove that (a + b)* = (a* + b*)*

Grammars

A grammar is a 4-tuple $G = \langle T, N, S, P \rangle$ where

- T is a finite set of terminals
- N is a finite set of non-terminals $\quad (N \cap T = \emptyset)$
- $S \in N$ starting symbol
- $P \subseteq (T \cup N)^* \times (T \cup N)^*$ s.t.

$$(u, v) \in P \implies \exists X \in N,\ l, z \in (T \cup N)^* : u = l X z$$

$(u, v_1), \ldots, (u, v_n) \in P$

$u ::= v_1 \mid \ldots \mid v_n$

$\longrightarrow$

$L(G) = \{ w \in T^* \mid S \Longrightarrow^* w \}$ where $\Longrightarrow\ = \{ (l u z, l v z) \mid l, z \in (T \cup N)^* \wedge (u, v) \in P \}$

Exercise 6
Find two derivations of the regular expression 1 + a . b using the grammar of regular expressions on page 16.

Term Algebras & Structural Induction

$\mathcal{V} \cap C = \emptyset$

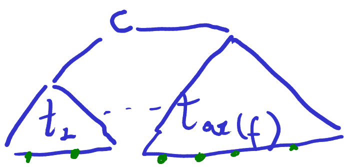
A signature $\Sigma = (C, ar)$ where $\begin{cases} C = \{c_1, \dots, c_n\} \\ ar : C \to \omega \end{cases}$

Term Algebra The term algebra on a signature Σ and a countable set $\mathcal{V}$ of variables is the smallest set $\mathrm{Term}_{\Sigma,\mathcal{V}}$ s.t.

- $\mathcal{V} \in \mathrm{Term}_{\Sigma,\mathcal{V}}$
- $\forall c \in C,\ t_1, \dots, t_{ar(t)} \in \mathrm{Term}_{\Sigma,\mathcal{V}} : c(t_1, \dots, t_{ar(f)}) \in \mathrm{Term}_{\Sigma,\mathcal{V}}$

$T_\Sigma = \mathrm{Term}_{\Sigma,\emptyset} \subseteq \mathrm{Term}_{\Sigma,\mathcal{V}}$ is the set of closed terms

Exercise 3
Explain why in the above definition it is essential to require that $\mathrm{Term}_{\Sigma,\mathcal{V}}$ is the smallest set



- are either variables or "constants" (i.e. $c \in \Sigma$ s.t. $ar(c) = 0$)

Exercise 4
Give the term algebra for regular expressions

Structural Induction (in general)

Given an inductively defined set, the structural induction principle can be used to prove properties on the elements of the set.

Principle of structural induction

a proposition

To prove $\forall t \in \mathrm{Term}_{\Sigma,\emptyset} \, . \, p(t)$

Base case(s) show $p(c)$ for all c s.t. $ar(c) = 0$

Inductive step show

$$p(t_1) \wedge \dots \wedge p(t_k) \Longrightarrow p(c(t_1, \dots t_k))$$

for all c s.t. $ar(c) = k > 0$ and $t_1, \dots, t_k \in \mathrm{Term}_{\Sigma,\emptyset}$

Exercise 5
Prove that, for every list l of natural numbers, sum(l) <= max(l)*len(l) where sum(l) is the sum of the elements of l, max(l) is the greateast element in l (assume max([]) = 0), and len(l) is the lenght of l

What do we mean by correctness?

Safety: "nothing bad happens"

Examples:

- if a number is printed, then 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

$Sys = (S, \rightarrow)$ where

- S is a set of states ← a.k.a. configurations
- $\rightarrow \subseteq S \times S$

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

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

Sys is deterministic if $\forall s, s', s'' \in S : s \rightarrow s' \wedge s \rightarrow s'' \Rightarrow s' = s''$

$$\begin{tikzcd} & s \arrow[dl] \arrow[dr] & \\ s' \arrow[rr, phantom, "="] & & s'' \end{tikzcd}$$

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.

Another (important) variant of TS

A **labelled transition system** is a triple $(S, A, \rightarrow)$ where

- S is a set of states
- A is a set of labels (or actions, or operations, or events, ...)
- $\rightarrow \subseteq S \times A \times S$ $\quad (\rightarrow : S \rightarrow 2^{A \times S})$ transition relation

$$s \xrightarrow{a} s'$$

observable information about what happens during the transition

particularly handy to model communication / concurrency / distribution

Example An FSA $M = (Q, \Sigma, q_0, \delta, F)$ is an LTS:

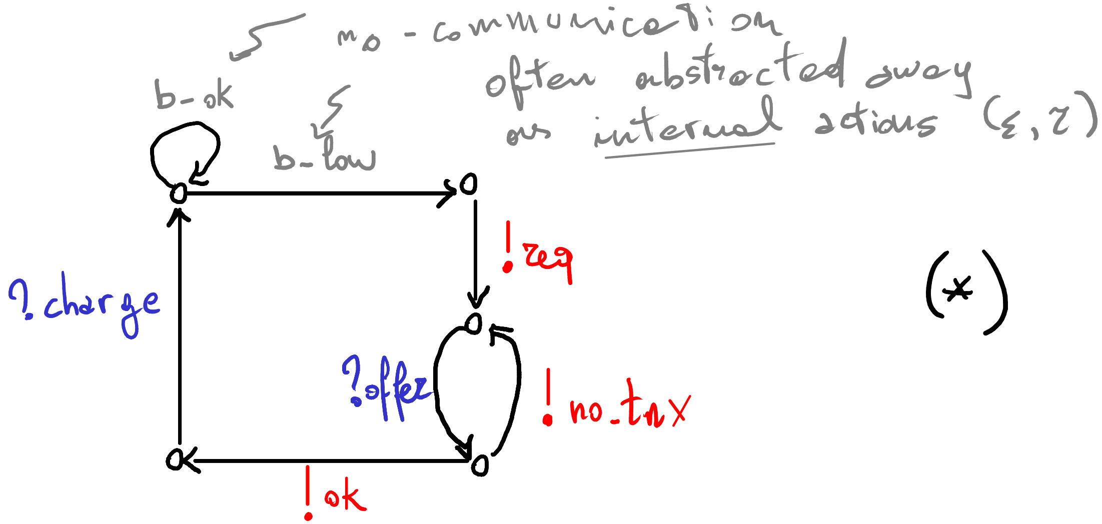
$LTS_M = (S \cup \{\bullet\}, \Sigma \cup \{\checkmark\}, \rightarrow)$ where $s \xrightarrow{a} s' \iff a \in \Sigma \;\&\; s' \in \delta(s,a)$ or $a = \checkmark \;\&\; s' = \bullet \;\&\; s \in F$

$$\mathcal{L}_M = \{ a_1 .. a_n \in \Sigma^* \mid \exists q_1, \dots, q_n \mid q_0 \xrightarrow{a_1} \dots q_{n-1} \xrightarrow{a_n} q_n \xrightarrow{\checkmark} \bullet \}$$

Communication-based concurrency

A robotic scenario:
Some mobile robots need to manage their energy in order to accoplish their task (e.g., patrolling some premises).
- When their batteries deplete, robots look for a recharge.
- Recharges are offered by recharge stations or other robots.

We can model this behaviour using an LTS capturing the observable features we are interested in: in this case communication
For instance, the behaviour of a robot seeking for a recharge is



The set of labels is the union of

- {b_low, b_ok} internal actions
- {?charge, ?offer} input actions
- {!req, !no_tnx} output actions

Exercise 4
Give an LTS modelling the behaviour of a robot offering a recharge.
Reflect about the "compatibility" between your solution and the LTS (*) above