From Singleton to Linear Logic
Frank Pfenning
OPLSS 2019

Date Performed: June 18th 2019
Students: J.W.N. Paulus
R. Gurdeep Singh
H. C. A. Tavante

3 Lecture 3: From Singleton to linear logic

3.1 A counter

We shall implement a counter with the following:

$$bin \;=\; \oplus\{b_0 : bin, b_1 : bin, e : \alpha\}$$

$$bin \;\vdash\; succ\, bin$$

$$\begin{aligned} succ = \textsc{case}\mathrm{L}(b_0 &\Rightarrow R.b_1; \leftrightarrow && (1)\\ |b_1 &\Rightarrow R.b_0; succ && (2)\\ |e &\Rightarrow R.b_1; R.e; \leftrightarrow) && (3) \end{aligned}$$

$$ctr = \&\{inc : ctr, reset : ctr, val : bin\} \qquad (4)$$

$$bin \;\vdash\; counter : ctr$$

$$\begin{aligned} counter = \textsc{case}\mathrm{R}(inc &\Rightarrow succ \overset{bin}{\mid} counter\\ |reset &\Rightarrow delete \overset{bin}{\mid} counter\\ |val &\Rightarrow \leftrightarrow) \end{aligned}$$

For delete:

$$bin \vdash delete : bin$$

$$\begin{aligned} delete = caseL(&b_0 \Rightarrow delete \\ &|b_1 \Rightarrow delete \\ &|e \Rightarrow R.e; \leftrightarrow) \end{aligned}$$

3.2 Turing machine

To show that the system at hand is Turing complete, we emulate a Turing machine (TM) (Wikipeda).
A Turing machine is a theoretical device with and infinite tape as memory. A head read and writes to this tape and can travel LEFT or RIGHT. The movement and actions of the head are encoded in a so called transition function δ.
The transition function δ is has following form:

$$\delta(\underbrace{q}_{\text{current state}}, \underbrace{a}_{\text{current tape data}}) = (\underbrace{b}_{\text{new tape contents}}, \underbrace{\begin{matrix}\text{LEFT}\\ \text{RIGHT}\end{matrix}}_{\text{movement}}, \underbrace{p}_{\text{next state}})$$

We will represent each state of our TM as a process that has tape data left and right of it. That is, the state is encoded as sitting in between to cells of the tape. For clarity, we give each state a name with a triangle ($\blacktriangleleft$ or $\blacktriangleright$) that indicates at which location on the tape the head in "looking".
The states of our Turing machine will be consuming data on the tape. Therefore, the tape must be emitting data, or in type terms, the tape should have an external choice type ($\oplus$). The tape left and right of the head should emit data in opposite directions. The tape data should be coming towards the processes. In our pictures we indicate the direction that tape content is being emitted in with an arrow. Let $\{a, b, \dots\}$ be the alphabet of the tape. The type of the tape left of the head is given by *tape*, the content on the right has type *etap*:

$$tape = \oplus\{a : tape, b : tape, \dots\} \qquad epat = \oplus\{a : tape, b : tape, \dots\}$$

For a state q of the Turing machine we wish to implement, we have two states in our logical system. One where the state "looks" left ($\blacktriangleleft\ q$) and one where the state "looks" right ($q\ \blacktriangleright$).

$$tape \vdash \blacktriangleleft\ q : epat$$
$$tape \vdash q\ \blacktriangleright : epat$$

To implement the TMs transition function, we need to define the above two processes for each state q in the TM. These processes should take into account the value of $\delta(q, \cdot)$.

a. For left moving rules ($\delta(q, a) = (b, \text{LEFT}, p)$) we have that:

- $\boxed{\ldots \mid \overrightarrow{a} \mid \blacktriangleleft\, q \mid \ldots}$ must become $\boxed{\ldots \mid \blacktriangleleft\, p \mid \overleftarrow{b} \mid \ldots}$

$$tape \vdash \blacktriangleleft\, q : epat$$
$$\blacktriangleleft\, q = \text{CASEL}(a \Rightarrow \blacktriangleleft\, p | (Lb; \leftrightarrow) \quad , b \Rightarrow \ldots)$$

- $\boxed{\ldots \mid q \blacktriangleright \mid \overleftarrow{a} \mid \ldots}$ must become $\boxed{\ldots \mid \blacktriangleleft\, p \mid \overleftarrow{b} \mid \ldots}$

$$tape \vdash q \blacktriangleright : epat$$
$$q \blacktriangleright = \text{CASER}(a \Rightarrow \blacktriangleleft\, p | (Lb; \leftrightarrow) \quad , b \Rightarrow \ldots)$$

b. For right moving rules ($\delta(q, a) = (b, \text{RIGHT}, p)$) we have that:

- $\boxed{\ldots \mid \overrightarrow{a} \mid \blacktriangleleft\, q \mid \ldots}$ must become $\boxed{\ldots \mid \overrightarrow{b} \mid q \blacktriangleright \mid \ldots}$

$$tape \vdash \blacktriangleleft\, q : epat$$
$$\blacktriangleleft\, q = \text{CASEL}(a \Rightarrow (Rb; \leftrightarrow) | q \blacktriangleright \quad , b \Rightarrow \ldots)$$

- $\boxed{\ldots \mid \overrightarrow{a} \mid q \blacktriangleright \mid \ldots}$ must become $\boxed{\ldots \mid \overrightarrow{b} \mid q \blacktriangleright \mid \ldots}$

$$tape \vdash \blacktriangleleft\, q : epat$$
$$\blacktriangleleft\, q = \text{CASER}(a \Rightarrow (Rb; \leftrightarrow) | q \blacktriangleright \quad , b \Rightarrow \ldots)$$

With this we have a translation scheme to translate any TM to our model. If s is the start state of the TM, we can use $s \blacktriangleright$ to emulate the TM. (One must still proof that the executions will yield the same result...)
There are two properties we wish to prove and these are topical properties you would find in a functional calculus.

Theorem 1 (Preservation) *if $A \vdash P : C$ and $P \to Q$ then $A \vdash Q : C$*

This theorem allows us to have the property that with every reduction step we are able to preserve the typing of the processes.

Theorem 2 (Progress) *if $A \vdash P : C$ then either:*

- $P \to Q$
- $P = \leftrightarrow$
- *P communicates left or right*

As there is no notation of values we have to define when a process has no reductions that can take place on them. The progress rule states that if a process is well typed then a reduction can be performed or P can no longer reduce and will either be a port forwarding or be waiting to send or receive a value.
We now wish to relax our context to allow multiple assumptions

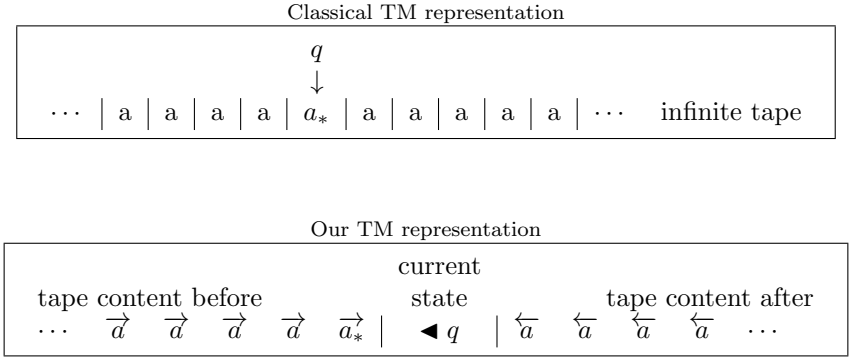

Figure 1: *Top*: A classical TM that is in state q and has a_* on the tape. *Bottom*:Visualization of a process "◀ q" on a tape. The Turing machines current cell under the head contains a_*

3.3 Example: Merge

We wish to describe a merging process that takes in two streams of bits and outputs the merge of these two streams. We will type this as: $x : bits, y : bits \vdash merge :: z : bits$
The define this process as:

$$z \leftarrow merge \leftarrow x\, y = case\, x\, (b_0 \Rightarrow z.b_0 \mid z \leftarrow merge \leftarrow y\, x \\ b_1 \Rightarrow z.b_1 \mid z \leftarrow merge \leftarrow y\, x)$$

$merge$ is written as $z \leftarrow merge \leftarrow x\, y$ because you can read $merge$ as a function that takes inputs x, y and then send its outputs along z. This merge function works by looking only at the bit that is sent along x and forwarding it along z then recursively calling itself however switching x and y meaning the next call reads the bit from y
We must now define a new context that allows multiple assumptions. We do this recursively as

$$\Gamma ::= \bullet \mid \Gamma, x : A$$

Our context is now either empty or contains a channel of type A and also consists of another context.
We must now rewrite our rules allowing us to have inputs across multiple channels. The first rule we shall rewrite is the $\oplus R_1$ and $\oplus R_2$ giving a more general case for more then two choices. The type of this will be $\oplus\{l : A_l\}_{l \in L}$ meaning label l in a set of labels L will provide type A_l

$$\frac{(k \in L) \quad \Gamma \vdash P :: x : A_k}{\Gamma \vdash x.k; P :: x : \oplus\{l : A_l\}_{l \in L}}\ \oplus R_k$$

In this rule $x.k$ is able to provide multiple types but is is only when the decision is made on x of k that we know which type is provided. Clearly we should be able to type the selection that has been made.

Type	Action	Continuation
$x : \oplus\{l : A_l\}_L$	send $k \in L$	$x : A_k$

x is able to provide a multitude of types. The sending of its label is when there is a synchronization deciding which type is being sent. x will then perform as the type agreed upon in the synchronization.

$$\frac{(\forall l \in L) \quad \Gamma, x : A_l \vdash Q_l :: z : C}{\Gamma, x : \oplus\{l : A_L\}_L \vdash case\, x\, (l \Rightarrow Q_l)_{l \in L} :: z : C}\ \oplus L$$

As expected if you are able to make multiple choices then we must also have multiple cases. One case for each choice. When x receives a case it performs as the process that the chosen case defines. Since we do not know what label will be provided then we must be able to type every outcome.

$$\frac{\Gamma_1 \vdash P :: x : A \quad \Gamma_2, x : A \vdash Q :: z : C}{\Gamma_1, \Gamma_2 \vdash (x \leftarrow P; Q) :: z : c}\ CUT_x$$

The cut rule allows processes P and Q to communicate across some channel x. Process P provides along x and x receives. Hence we have two contexts, one describing the channels of P and one the channels of Q. We must be able to prove that given the context for P that is is able to provide A across the connecting channel x and that given Q's context and also the A that P provides along x it is able to consume it and provide c across z

$$\frac{}{A \vdash A}\ ID \qquad \frac{}{y : A \vdash x \leftarrow y :: x : A}\ ID$$

We finally describe how the identity works. The idea is that what ever is received across y is just sent along x. We must only have y in our context at it is linear, everything must get used up.