2.5 Turing Machine Schemas

Specifying Turing Machines in monolithic fashion is cumbersome and complex. We introduce a compositional method for Turing Machine specification whereby:

- we introduce notation for representing whole Turing Machines, abstracting over their internal workings.
- a method for joining Turing Machines, so as to construct complex Turing Machines from simpler ones.

Specifying even relatively simple Turing Machines requires substantial notation. This problem is endemic to any model of programming not just Turing Machines, i.e., any programming language. In computer science, we usually tame such complexity through abstraction and compositionality, and Turing Machines provide the ideal setting where to showcase such modus operandi. We can organise the task of describing Turing Machines by adopting:

- a method of constructing complex Turing Machines from simpler ones;
- a simple notation for describing simple Turing Machines i.e., the building blocks for more complex machines

Let us begin with the building blocks on an arbitrary but fixed alphabet. These will consist of simple Turing Machines performing a single operation such as moving the head (left or right) or writing a particular symbol, and then halt. For instance the Turing Machine $L$ defined on Slide 38 (denoted by the circle labelled $L$) moves the head to the left and halts. The machine has two states, $q_0$ and $q_H$, and its transition function (schema\(^2\)) is defined as follows (where $x$ ranges over the symbols in $\Sigma$):

$$\delta(q_0, x) = (q_H, L)$$

We can define an analogous machine that moves the head right instead, and call it $R$. Another simple kind of Turing Machine (also defined in on Slide 38) is called $a$, and it simply writes the symbol $a$ at the current location of the head and halts. Again, it has just two states and a transition (schema):

$$\delta(q_0, x) = (q_H, a)$$

Slide 38 also defined a special kind of Turing Machine we denote using an empty square. As we shall see, this machine acts as the identity when composed with other machines, i.e., a skip in programming language jargon, but more specifically we will often use it as the terminal Turing Machine at the level of Turing Machine schemas. Hereafter we refer to this Turing Machine as $M_{halt}$.

Before embarking on the formal definitions of a Turing Machine Schema, let us first consider how arbitrary Turing Machines can be “connected” together in order to make more complex machines. For instance, given two Turing Machines $M_1$ and $M_2$ over the same alphabet, $\Sigma$, but with different disjoint sets of states, $Q_1 \cap Q_2 = \emptyset$, we derive the (composite) Turing Machine $M_1 \xrightarrow{a} M_2, M_1 \xrightarrow{\#} M_1$ (where $\Sigma = \{a, \#\}$) as follows:

\(^2\)We call this a transition schema because it denotes a template for generating a number of transitions, depending on $\Sigma$. Do not confuse transition schemas with Turing Machine schemas.

\(^3\)Recall from previous courses that we can always ensure this state separation by renaming conflicting states.
Building Block Schema Examples

Slide 38

\(M_1 \xrightarrow{a} M_2\): we add the transition from the final state of \(M_1\) to the initial state of \(M_2\) i.e.,

\[\delta(q_1^1, a) = (q_0^2, a)\]

Note that the transition is in some sense an identity transition wrt. the tape as it leaves its contents unchanged. This added transition also has the side effect of transforming the final state in \(M_1\) to a normal state as, by Def. 1, final states cannot have any transitions from them.

\(M_1 \xrightarrow{\#} M_1\): we add a “loop-back” transition from the final state of \(M_1\) to the initial state of \(M_1\) itself.

\[\delta(q_1^1, \#) = (q_0^1, \#)\]

This transition "re-launches" \(M_1\) while leaving the contents of the tape unchanged.

From an abstract level, these high-level transitions can also be seen as branching transitions, deciding which machine to launch next depending on the current tape contents at the head location, resulting from the sub-computation just completed. This informal description is represented diagrammatically on Slide 40.

There is one final detail left unspecified. Recall that we implicitly changed the final state of \(M_1\) to a normal state, thereby leaving only one final state, that of \(M_2\) to be the only candidate for the final state of the composite machine. However, we have not specified which of the two start states \(q_0^1\) and \(q_0^2\) of \(M_1\) and \(M_2\) respectively, will act as the start state for the composite machine (recall that we can only have one start state for any Turing Machine). By convention, when this is not stated explicitly, we shall often assume that the composite machine start state is the start state of the first indexed sub-machine i.e., in this case \(M_1\). This assumption means that we also transform \(q_0^2\) to a normal state. In what follows though, we will specifically identify the start Turing Machine (containing the start state of the composite machine).

More formally, we describe how a finite collection of Turing Machines can be connect in this manner as in Slide 41. A machine schema represents a composite Turing Machine as defined in Def. 27 on Slide 42.

We shall use a particular notation to describe the schema. We shall write graphs denoting schemas as follows, describing both the set \(M\) as well as the function \(\eta\):
Composing Turing Machine as Schemas (Informal)

For arbitrary $M_1 = (Q_1, \Sigma_1, \delta_1)$ and $M_2 = (Q_2, \Sigma_2, \delta_2)$ where ($\Sigma_1$ and $\Sigma_2$ typically coincide and):

- $Q_1 \cap Q_2 = \emptyset$;
- $Q_1 = \{q_0^1, q_1^1, q_2^1, \ldots\}$ and $Q_2 = \{q_0^2, q_1^2, q_2^2, \ldots\}$;

The composite machine description

$M_1 \xrightarrow{a} M_2$, $M_1 \xrightarrow{\#} M_1$

denotes $M_3 = (Q_3, \Sigma_3, \delta_3)$:

$Q_3 = Q_1 \cup Q_2$
$\Sigma_3 = \Sigma_1 \cup \Sigma_2$
$\delta_3 = \delta_1 \cup \delta_2 \cup \{(q_H, a) \mapsto (q_0^3, a), (q_H, \#) \mapsto (q_0^3, \#)\}$

Slide 39

- $M_0$ is identified by the node marked with an edge that has no source node.
- The graph contains exactly one halting node, represented by a square node and the Turing Machine at this node will be $M_{halt}$.
- An edge from $M_i$ to $M_j$ is labelled $a$ iff $\eta(M_i, a) = M_j$; we sometimes write such connections as $M_i \xrightarrow{a} M_j$.
- An edge from $M_i$ to $M_j$ is unlabelled iff $\eta(M_i, a) = M_j$ for all $a \in \Sigma$; we sometimes write such connections as $M_i \xrightarrow{\_} M_j$, coalescing the nodes as one; The node $M^k$ denotes the concatenation of $M$ for $k$ times $M \ldots M$.
- An edge from $M_i$ to $M_j$ is labelled $\bar{a}$ iff $\eta(M_i, b) = M_j$ for all $b \in \Sigma$ such that $b \neq a$; we sometimes write such connections as $M_i \xrightarrow{\bar{a}} M_j$.
- Nodes marked $L$, $R$, and $a$ (for $a \in \Sigma$) represent basic Turing Machines described earlier on Slide 38.

Example 28 (Expressing Turing Machines as Schemas). As an example, consider our earlier Turing Machine $M_{erase}$ described on Slide 12 with the diagram on Slide 8. Recall that this machine simply erased all the non-blank symbols on the tape. Slide 43 shows how we can derive the same Turing Machine using simple Turing Machine building blocks and schemas to connect them together into a composite Turing Machine. Note that including $M_{halt}$ in $M$ means that it is present as two distinct nodes in the graph of Slide 43 i.e., an ordinary node and the final halting node. As we stated earlier, as the ordinary node, $M_{halt}$ acts as the no-op skip.

After showing how to structurally construct complex Turing Machines from simpler ones, we now turn our attention to the behaviour of the composite Turing Machine that can be inferred from the behaviour of the component Turing Machines in the schema. Central to attaining such an inference is Lemma 29 on Slide 44 which implies that if the machines composed are $M_1$ and $M_2$ by some transition $a$, the behaviour...
of their composite machine on an input \( x \) behaves first as \( M_1 \) on \( x \) and then as \( M_2 \) on the output of \( M_1 \) as its input provided that the head reads \( a \) at the halting of \( M_1 \). More specifically, the Lemma states that if a computation occurs with certain (minimal) contents on the tape, then attaching additional tape on the left does not inhibit that particular computation form happening (once the head is suitably shifted to reflect the added tape piece).

(Proof for Lemma 29). We give a rigorous justification as to why the lemma should hold. From the assumption that \( \langle q_2, x_2a_2y_2 \rangle \vdash_M^\prec \langle q_3, x_3a_3y_3 \rangle \) we conclude that \( M \) did not hang during this computation, which also means that the head did not attempt to travel beyond the leftmost location on the substring \( x_2 \) in the configuration \( \langle q_2, x_2a_2y_2 \rangle \) (otherwise the machine would have crashed, thus hanged). Thus, starting from \( q_2 \), following exactly the same sequence of steps on the extended tape \( wx_2a_2y_2 \) we are guaranteed that the head never travels to the part of the tape containing \( w \), leaving \( w \) unchanged. Thus we should conclude that \( \langle q_2, wx_2a_2y_2 \rangle \vdash_M^\prec \langle q_3, wx_3a_3y_3 \rangle \) and then by \( \langle q_1, x_1a_1y_1 \rangle \vdash_M^\prec \langle q_2, wx_2a_2y_2 \rangle \) and transitivity we obtain \( \langle q_1, x_1a_1y_1 \rangle \vdash_M^\prec \langle q_3, wx_3a_3y_3 \rangle \). □

From this we can conclude that the computation (behaviour) of a composite machine is obtained as a straightforward chaining of the sub-computation of the simpler Turing Machines, chosen at the end of the preceding computation depending on the contents pointed to by the head. Perhaps closer to the way we are accustomed to program, through schemas Turing Machine behaviour can be envisaged as the sequence of \( n \) subroutine calls operating on different parts of the tape in sequence and determining which subroutine to call next. Schemas allow us to use Turing Machines as our basis for inching towards what is effectively a very rudimentary programming language.

2.5.1 Exercises

1. Describe a Turing Machine recognising palindromes over \( \{a, b\} \) using schemas and the following machines:
   - \( \gg, \ll \) which move the head to the rightmost (resp. leftmost) non-blank symbol in a string.
   - \# which writes a "#" symbol and terminates.
Turing Machine Schema Definition

**Definition 26** (Machine Schema). A machine schema is a triple $\langle M, \eta, M_0 \rangle$ where

- $M$ is a *finite set* of Turing Machines sharing a common alphabet $\Sigma$ but having mutually disjoint state sets;
- $\eta$ is a partial function $M \times \Sigma \rightarrow (M \cup \{M_{\text{halt}}\})$;
- $M_0$ is the initial machine, whereby $M_0 \in M$.

and $M_{\text{halt}}$ is the Turing Machine that halts upon every input (after rewriting the same input on tape).

---

2. Express the Turing Machine recognising the language $a^n b^n c^n$ in terms of Turing Machine schemas. If possible reuse as many sub-Turing Machines already defined as possible in your schema.

3. Recall $L_1 = \{a^n \mid \exists k. f^k(n) = 1\}$ where:

$$f(n) \overset{\text{def}}{=} \begin{cases} n/2 & \text{if } n \text{ is even} \\ 3n + 1 & \text{otherwise} \end{cases}$$

The algorithm for recognising language $L_1$ was defined as:

(a) if the argument is equal to 1 then transition to the final state $q_H$ and accept.
(b) if the argument is even then
   i. divide by 2.
   ii. goto step 1.
else
   i. multiply by 3 and add 1.
   ii. goto step 1.

Express this algorithm as a Turing Machine schema.
**Turing Machine described by a Schema**

**Definition 27** (Schema to Turing Machine). A machine schema \( \langle M, \eta, M_0 \rangle \) represents the Turing Machine \( \langle Q, \Sigma, \delta, q_0^0, \#, q_{halt}^0 \rangle \) where:

\[
Q \overset{\text{def}}{=} \bigcup_{i=0}^{n} (Q_i \cup q_{halt}^i) \cup q_0^halt
\]

\[
\delta(q, a) \overset{\text{def}}{=} \begin{cases} 
\delta_i(q, a) & \text{if } q \in Q_i \\
(q_0^i, a) & \text{if } q = q_{halt}^i \text{ and } \eta(M_i, a) = M_j
\end{cases}
\]

where \( M = M_0, \ldots, M_n \) and each \( M_i = \langle Q_i, \Sigma, \delta_i, q_0^i, \#, q_{halt}^i \rangle \)

---

**Expressing \( M_{erase} \) as a schema**

![Diagram](attachment:image.png)

---

**Slide 42**

**Slide 43**

31
Lemma 29. If for any Turing Machine $M$ we have

$$\langle q_1, x_1a_1y_1 \rangle \xrightarrow{*}_M \langle q_2, wx_2a_2y_2 \rangle$$

and

$$\langle q_2, x_2a_2y_2 \rangle \xrightarrow{*}_M \langle q_3, x_3a_3y_3 \rangle$$

then this implies

$$\langle q_1, x_1a_1y_1 \rangle \xrightarrow{*}_M \langle q_3, wx_3a_3y_3 \rangle$$