1 How are transducers encoded?

Definition 1.1. Let (Q, q_0, V, B, E, S, T) be a stateful transducer. Q is a finite set of states, and q_0 is the initial state of the stateful transducer. V is a finite set of registers, each a finite width bit string. We can index V as $V_1, V_2, \ldots, V_n$, and we say that width(V_i) is the width of register V_i. B is a language of bit-operator arithmetic with the usual suspects: $\{+, *, <<, >>, -, \ldots\}$ as operators, and a base set of values a set of registers and set of bit-string integers, without being to careful, the language B is defined such that operators can only be applied to operands with consistent bit-widths (I can make this better). We can select a subset B_x of B, meaning the subset of expressions in B that evaluate to values of x width. S is a finite set of input language characters. T is a finite set of output language characters. E is a set of six tuples $(Q, S, B_1, 2^{(V \leftarrow B_{\text{width}(v)})}, T, Q)$ representing transitions. The elements are respectively: the start node of the edge, the accepting input character, an additional Boolean guard, a set of assignment to the registers from Boolean expressions such that widths checkout, the output character, and the target node of the edge. Note there is no final accepting set of nodes.

The idea of the encoding is to describe the machines logic using a set of unbound variables, for the input, output, register start state, register end state, start node state, and end node states. Call these variables the timeless variables. These variables exclusively are used to describe the logic of the edges. Then we create a set of copies of variables for the input, output, register state, and node state for each of some finite n number of steps. Call these variables the timeful variables. We then sow together the timeless variables and the timeful variables between each pair of time steps (i and $i + 1$). We do this by saying there exists timeless variables such that everything makes sense with the timeful variables (e.g. the input timeless variable is the input variable for time step i, and the register end state is the register state for time step $i + 1$), and that the edge logic described using timeless variables is true. We then conjoin all these existential statements together. We provide a written description as well as judgements in Figure **??**.

1.1 Basic Conversions

First we will discuss the encoding of some constant values. Q, S, and T are encoded by using some bijection between those sets and the natural numbers up to their magnitude. Let $\sharp Q$, $\sharp S$, and $\sharp T$ be encodings for each set respectively. $\sharp q_0$, the encoding of $\sharp q_0$, is just $(\sharp Q)(q_0)$.

1.2 Encoding Registers

We use two variables for some register v_i: $\sharp(v_i)$ and $\sharp(v_i)'$. The former being a variable representing input values and the latter output values.

1.3 Encoding bit-operator arithmetic

Bit-operator arithmetic has a very natural encoding by just mapping each operator to it's respect first order logic encoding. Any use of a register is simply encoded as $\sharp(v_i)$, that is the input register value.

1.4 Encoding assignment

For some assignment $v_i \leftarrow b$, we have $\sharp(v_i \leftarrow b) := \sharp(v_i)' = \sharp(b)$.

1.5 Encoding edges

For some edge (q, s, g, a, t, q'), we have the encoding:

$$\sharp(q, s, g, a, t, q') := (m_s = \sharp q \wedge m_i = \sharp s \wedge \sharp g) \implies (\wedge \sharp a \wedge m_o = \sharp t \wedge m_t = \sharp q')$$

m_s, m_i, m_o, and m_t are variables representing the start state, input character, output character, and target state respectively. They will be bound, as shown in the next section, as existential variables set equal to time time dependent states and characters. We then create an encoding of all edges by conjoining each of these implications together. We also add a special default annotation such that if $\neg(m_s = \sharp q \wedge m_i = \sharp s)$ for paired q and s, the implication is false.

1.6 Encoding the machine

The machine's logic is encoded up to some fixed time n, meaning that the encoding will correctly execute the logic for n input and output characters, with any extra output being undefined behavior. For each $0 \leq i < n$, we create the following variables: a state variable s, an input variable i, an output variable o, a variable for each register $w_{1\ldots n}$. We will refer to each of these set of variables as time slice. We then encode the transfer from time i to $i + 1$ as:

$$\begin{aligned} &\exists m_s, m_i, m_o, m_t, v_1, v_2, \ldots, v_n, v_1', v_2', \ldots, v_n'. \\ &m_s = s_i \wedge \\ &m_i = i_i \wedge m_o = o_i \wedge m_t = s_{i+1} \wedge \\ &v_k = (w_k)_i \wedge v_k' = (w_k)_{i+1} \wedge \text{encoding of all the edges as given in the previous} \end{aligned}$$

Also, we set $(w_k)_0$ to appropriate initial register values, and $s_i = \sharp(q_0)$ that is to the start node.

1.7 Encoding examples

Finally, for each input/output example given we encode a machine up to the length of the input output example, and then take the conjunction of each copy.

$$\text{REGISTER}\ \frac{}{\sharp(v_i) := v_i}$$

$$\text{OPERATOR}\ \frac{A := \sharp(a) \qquad B := \sharp(b) \qquad \otimes := \sharp(\oplus)}{\sharp(a \oplus b) := A \otimes B}$$

$$\text{ASSIGNMENT}\ \frac{B := \sharp(b)}{\sharp(a \leftarrow b) := a' = B}$$

$$\text{EDGE}\ \frac{\begin{array}{c} Q := \sharp(q) \\ S := \sharp(s) \qquad G := \sharp(g) \qquad A := \sharp(a) \qquad T := \sharp(t) \qquad Q' := \sharp(q')) \end{array}}{\sharp((q, s, g, a, t, q')) := (m_s = Q \wedge m_i = S \wedge G) \implies (A \wedge m_o = T \wedge m_t = Q')}$$

$$\text{EDGES}\ \frac{E_i := \sharp(e_i)}{\sharp(\{e_1, e_2, \ldots, e_n\}) = (\text{not any other edge}) \bigwedge_{i \in [1,n]} \sharp(e_i)}$$

$$\text{STEP}\ \frac{E_e := \sharp(E)}{\begin{array}{c} \sharp(\text{step}(i, i+1)) := \exists m_s, m_i, m_o, m_t, V, V'. E_e \wedge m_s = s_i \wedge m_i = i_i \wedge m_o = o_i \\ \wedge m_t = s_{i+1} \wedge v_k = (w_k)_i \wedge v'_k = (w_k)_{i+1} \end{array}}$$

$$\text{EXAMPLE}\ \frac{i = \text{input string} \wedge o = \text{output string} \wedge n = |i| = |o|}{\exists_{i<n} i_i, (w_k)_i, s_i, o_i. \text{initial register states set} \wedge_{i<n} \sharp\text{step}(i, i+1)}$$