Constructive Logic (15-317), Spring 2022
Recitation 14: Session Types and Review

Recall from lecture that we translated a finite-state transducer which compressed runs of b into single instances of b into an ordered program in two different ways — first as an ordered forward logic program, then as a concurrent program in the subsingleton fragment of ordered logic. We will look here at some further examples of this type.

Task 1. Write a transducer over the alphabet a, b which produces ab for every occurrence of ab in the input and erases all other symbols.

1. Present it in the form of ordered inference rules (for a forward ordered logic program).
2. Present it in the form of a well-typed concurrent program in the subsingleton fragment of ordered logic.

Solution 1: Reading strings from left-to-right, with an end-of-string symbol \$. e.g. this program can be run on the string $babb$ by starting in the state $q_0\ b\ a\ b\ b\ \$$:

$$\frac{q_0\ a}{q_1} \qquad \frac{q_0\ b}{q_0} \qquad \frac{q_0\ \$}{\$}$$

$$\frac{q_1\ a}{q_0} \qquad \frac{q_1\ b}{a\ b\ q_0} \qquad \frac{q_1\ \$}{\$}$$

If we instead read strings from right to left, starting in the state $\$\ b\ a\ b\ b\ q_0$ for the string $babb$, we instead can give the following forward ordered logic program:

$$\frac{a\ q_0}{q_0} \qquad \frac{b\ q_0}{q_1} \qquad \frac{\$\ q_0}{\$}$$

$$\frac{a\ q_1}{q_0\ a\ b} \qquad \frac{b\ q_1}{q_0} \qquad \frac{\$\ q_1}{\$}$$

To implement this transducer as a concurrent program, we define a process for each state. In this example, we will read the string from right-to-left, as in the second forward logic program above. It is, of course, possible to present a program that reads the string from left to right, instead, although this also means changing how exactly a string is represented as a process (or as a sequence of messages).

$string \vdash Q_0 : string$
$Q_0 = \mathsf{caseL}\ (a \Rightarrow Q_0$
$\qquad\qquad \mid b \Rightarrow Q_1$
$\qquad\qquad \mid \$ \Rightarrow \mathsf{R}.\$\ ;\ \leftrightarrow$
$\qquad\qquad)$

$string \vdash Q_1 : string$
$Q_1 = \mathsf{caseL}\ (a \Rightarrow \mathsf{R}.b\ ;$

$\mathsf{R}.a$;
Q_0
$\mid b \Rightarrow Q_0$
$\mid \$ \Rightarrow \mathsf{R}.\$$; $\leftrightarrow$
)

Task 2. Reconsider the transducers for compressing runs of b's, given here as a set of ordered inference rules. We present here the version without an explicit final state.

$$\frac{a\ q_0}{q_0\ a} \qquad \frac{b\ q_0}{q_1\ b} \qquad \frac{\$\ q_0}{\$}$$

$$\frac{a\ q_1}{q_0\ a} \qquad \frac{b\ q_1}{q_1} \qquad \frac{\$\ q_1}{\$}$$

In our encoding as a program Q_0 of type $\mathit{string} \vdash Q_0 : \mathit{string}$ we treated letters as messages and states as processes. No explicit representation of the final state is necessary with the rules above.

Define a dual encoding where symbols of the alphabet and endmarkers are represented processes and states as messages.

1. Define an appropriate type *state* so that $\mathit{state} \vdash P_a : \mathit{state}$ where P_a is the process representation for the alphabet symbol a.
2. For each symbol a of the transducer alphabet, define the process P_a.
3. Give the type of the process $P_\$$ representing the endmarker \$. It may make sense to represent a final state with an explicit message, but you may also find it simpler not to.
4. Define the process $P_\$$ for the endmarker.
5. Define the initial configuration for the string *babb* and initial state q_0.
6. Describe the final configuration for the given example string and initial state (once the program has run to completion).
7. Consider how to compose two transducers encoded in this form. How does this compare to the composition of transducers in the original encoding given in lecture (via cut)?

Solution 2: Note that this is one solution, but that others may be possible.

1. $\mathit{state} = \&\{q_0 : \mathit{state}, q_1 : \mathit{state}\}$. You may notice that this type has no base case — it represents an infinite stream of states. This will simplify composing transducers, but again, is far from the only solution.
2. $\mathit{state} \vdash P_a : \mathit{state}$
   $P_a = \mathsf{caseR}\ (q_0 \Rightarrow \mathsf{L}.q_0\ ;\ P_a$
   $\qquad\qquad \mid q_1 \Rightarrow \mathsf{L}.q_0\ ;\ P_a$
   $\qquad\qquad)$

   $\mathit{state} \vdash P_b : \mathit{state}$

$P_b = \mathsf{caseR}\ (q_0 \Rightarrow \mathsf{L}.q_1\ ;\ P_b$
$\quad\quad\quad\quad \mid q_1 \Rightarrow \mathsf{L}.q_1\ ;\ \leftrightarrow$
$\quad\quad\quad\quad)$

3. $state \vdash P_\$: state$
   $P_\$ = \mathsf{caseR}\ (q_0 \Rightarrow P_\$$
   $\quad\quad\quad\quad \mid q_1 \Rightarrow P_\$$
   $\quad\quad\quad\quad)$

4. $P_\$\ P_b\ P_a\ P_b\ P_b\ (\mathsf{L}.q_0\ ;\ \leftrightarrow)$

5. $P_\$\ P_b\ P_a\ P_b$

6. To compose transducers in this form is a fairly involved task — the state type needs to be modified to account for both transducers' states, and each process P_x needs to handle both types of state. Once this is done, we can simply send a sequence of the initial states of the transducers to the collection of processes representing the string, which is relatively easy, but there is a substantial amount of work involved in modifying all of the processes for symbols.

   The approach used in lecture, by contrast, composes with minimal work — the processes for states for a transducer can remain the same, the type of strings can remain the same, and all we need to do is cut two processes together, feeding the output of one as input to the other.

1 Review

You should ask any questions you may have about the material of this course, either as preparation for the final, or for general interest. Below is a high-level list of the topics covered in the course, which you might expect to see come up to some extent on the final exam.

1. Proof Theory:
   - Natural Deduction
   - Harmony
   - Verifications and Uses
   - Proof terms
   - Sequent Calculus
   - Cut and Identity
   - Proof normalization/cut elimination
2. Proof Search:
   - Reduced Sequent Calculus
   - Inversion and the inversion calculus G4IP
   - Logic Programming

- Prolog
- Backwards Chaining
- Forwards Chaining and Forwards Logic Programming
- Focusing

3. Other logics and extensions to constructive logic:

    - Quantifiers
    - Heyting Arithmetic
    - Classical Logic
    - Modal Logic
    - Linear Logic
    - Ordered Logic
    - Ordered/Linear Logic proofs as concurrent programs
    - (Forward) Ordered/Linear Logic Programs