An Intensional Concurrent Faithful Encoding of Turing Machines

Thomas Given-Wilson

An Intensional Concurrent Faithful Encoding of Turing Machines

Thomas Given-Wilson
INRIA, Paris, France *
thomas.given-wilson@inria.fr

The benchmark for computation is typically given as Turing computability; the ability for a computation to be performed by a Turing Machine. Many languages exploit (indirect) encodings of Turing Machines to demonstrate their ability to support arbitrary computation. However, these encodings are usually by simulating the entire Turing Machine within the language, or by encoding a language that does an encoding or simulation itself. This second category is typical for process calculi that show an encoding of λ-calculus (often with restrictions) that in turn simulates a Turing Machine. Such approaches lead to indirect encodings of Turing Machines that are complex, unclear, and only weakly equivalent after computation. This paper presents an approach to encoding Turing Machines into intensional process calculi that is faithful, reduction preserving, and structurally equivalent. The encoding is demonstrated in a simple asymmetric concurrent pattern calculus before generalised to simplify infinite terms, and to show encodings into Concurrent Pattern Calculus and Psi Calculi.

1 Introduction

The benchmark for computation is typically given as Turing computability [24, 14, 15, 5]; the ability for a computation to be performed by a Turing Machine [23]. This choice of benchmark is also widely supported by various appeals to calculation of a "computable function" or "decidable predicate" or "recursive function" by a Turing Machine [24, 14, 7, 8, 13]. Indeed, since Turing [24], Kleene [14], Curry [7, 8] and others showed that Turing Machines can encode λ-calculus, general recursive functions, and combinatory logic [21], respectively, any language that can encode any of these can be considered to be able to do computation. However, these rely upon their encoding of Turing Machines which typically involve the simulation of a Turing Machine within the other language.

The typical simulation of a Turing Machine, say in λ-calculus, is to represent the tape as a list, and the symbols by natural numbers using Gödelisation. The operations of the Turing Machine are then handled by a function that can operate over the list (encoded tape) and compare the numbers using their Church encodings [2]. While such encodings preserve computation they have some weaknesses. The encoded computation takes many more reductions to produce the same operations; recognising a symbol requires a predecessor function, testing for zero, and then switching on numbers to determine the next symbol to write, all before reconstructing the list (encoded tape). Such encodings are not very clear; the representation of a symbol s may be mapped to some number i that is then represented as a function that is the ith iterator. These kinds of encodings are also metamathematical [22] in nature and so are always at least one level of mathematics away from the computation itself, which can lead to misunderstandings about the true expressiveness of a language [13].

Process calculi are often considered to generalise the sequential (non-concurrent) computation of λ-calculus by some form of encoding [16, 4, 18, 19, 6, 17, 20]. These encodings again have weaknesses

*This work has been supported by the project ANR-12-IS02-001 PACE.

Submitted to:
ICE 2014

such as adding reductions steps, lacking clarity, or even limiting reductions strategies (such as in Milner's encoding of λ-calculus into π-calculus [16], which is then built upon by those who use encoding π-calculus to capture computation). Further, these encodings are often up to some weak behavioural equivalence and can create many dead processes as a side effect. Thus a Turing Machine can be encoded into λ-calculus and then encoded into π-calculus and then encoded into another process calculus so that the original computation is now buried three levels of meta-operations deep, with almost no obvious connection to the original Turing Machine, and only weakly behaviourally equivalent to an encoding of the Turing Machine after the computation.

This paper attempts to avoid the worst of these various encoding issues by presenting a straightforward approach to encoding a Turing Machine into any process calculus that supports *intensional* communication [9, 10]. Intensionality is the capability for functions or communication primitives to operate based upon the internal structure of terms that are arguments or being communicated, respectively [13, 9]. Some recent process calculi support intensionality, in particular Concurrent Pattern Calculus [11, 12] and Psi Calculi [3]. The presentation here will use a simplified *asymmetric concurrent pattern calculus* (ACPC) to detail the translation as clearly as possible.

The intensionality of ACPC is an advanced form of pattern-matching that allows *compound* structures of the form $s \bullet t$ to be bound to a single name, or to have their structure and components be matched in communication. For example, consider the following processes:

$$P \stackrel{\text{def}}{=} \overline{a \bullet b} \to \mathbf{0} \qquad Q \stackrel{\text{def}}{=} \lambda x \bullet \lambda y \to Q' \qquad R \stackrel{\text{def}}{=} \lambda z \to R' \qquad S \stackrel{\text{def}}{=} a \bullet b \to S'$$

where P is an output of the compound $a \bullet b$. The inputs of Q and R have binding names of the form λx in their patterns $\lambda x \bullet \lambda y$ and λz, respectively. The input of S tests the names a and b for equality and performs no binding. These process can be combined to form three possible reductions:

$$\begin{aligned} P \mid Q &\longmapsto \{a/x, b/y\}Q' \\ P \mid R &\longmapsto \{a \bullet b/z\}R' \\ P \mid S &\longmapsto S' \,. \end{aligned}$$

The first matches the structure of the output of P with the input of Q and binds a and b to x and y, respectively, in Q'. The second binds the entire output of P to the single name z in R'. The third matches the structure and names of the output of P with the structure and names of the input of S, as they match they interact although no binding or substitution occurs.

The encoding presented here exploits the ability to represent symbols and structures into the output of a process, and to test for structure, equality, and to bind in an input to clearly represent a Turing Machine. Indeed, the encoding is *faithful* in that each operation that is performed by a Turing Machine yields exactly one reduction in the encoding to the (structurally) equivalent encoded Turing Machine after the operation. The key to the elegance of the encoding is to represent the current state q and tape $\mathcal{T}$ of the Turing Machine by an output of the form

$$\overline{q \bullet [\![\mathcal{T}]\!]} \to \mathbf{0}$$

where $[\![\mathcal{T}]\!]$ converts the tape to a convenient format. The transitions functions of the Turing Machine are then each encoded into a process of the form

$$q_i \bullet p \to \overline{q_j \bullet t} \to \mathbf{0}$$

where q_i and p match the current state, current symbol, and the structure of the tape, and the output exhibits the new state q_j and modified representation of the tape t. These transition functions can then be combined via parallel composition and replication in a manner that allows for a faithful encoding of a Turing Machine.

The elegance of this encoding can be built upon by considering some variations. It is straightforward to modify the encoding so that a tape with infinite blank symbols at the edges can be represented by a finite term in the output. Both of these encodings can then be easily captured by both Concurrent Pattern Calculus and Psi Calculi with only minor changes to the encoding and proofs.

There are two limitations for non-intensional process calculi. First is the inability to match multiple names in a single reduction. This can be worked around by encodings or match-rules that rely upon structural equivalence, however at some cost to the elegance of the encoding. Second proves impossible to fix; the inability for non-intensional calculi to bind an arbitrarily complex structure to a single name and still access the components. Thus alternative approaches must be used to encode Turing Machines into, say, synchronous polyadic π-calculus, at the cost of faithfulness and easy equivalence of encodings.

The structure of the paper is as follows. Section 2 recalls Turing Machines. Section 3 presents intensionality via asymmetric concurrent pattern calculus. Section 4 defines the encoding of Turing Machines in asymmetric concurrent pattern calculus. Section 5 considers variations on the encoding, including into published calculi. Section 6 discusses non-faithful encodings into, and limitations of other calculi. Section 7 draws conclusions.

2 Turing Machines

Each Turing Machine is defined by an alphabet, a set of states, a transition function, and a tape. In addition during operation the current head position and current state must also be accounted for. The alphabet $\mathcal{S}$ is the set of symbols s recognised by the Turing Machine and includes a special *blank* symbol b. The set of states $\mathcal{Q}$ is a collection of states q that the Turing Machine can transition to and from, and includes a *start state* q_0. The transition function $\mathcal{F}$ is represented by tuples of the form $\langle q_1, s_2, s_3, d_4, q_5\rangle$ that instructs the machine when the current state is q_1 and the current head position symbol is s_2 to write (to the current head position) s_3 and then move the current head position direction d_4 (either L for left or R for right), and change the current state to q_5. The *tape* $\mathcal{T}$ is an infinite sequence of cells each of which contains a symbol, this is denoted by $[\ldots b, s_1, s_2, s_1, b\ldots]$ that indicates an infinite sequence of blanks, the symbols s_1 then s_2 then s_1 and then an infinite sequence of blanks. The current head position can be represented by marking the pointed to symbol in bold, e.g. the tape $[\ldots b, \mathbf{s_1}, s_2, s_1, b\ldots]$ indicates that the current head position is the leftmost instance of s_1. Thus the definition of a Turing Machine can be given by $\langle\langle \mathcal{S}, \mathcal{Q}, \mathcal{F}, \mathcal{T}, q_i\rangle\rangle$ where q_i is the current state.

For a Turing Machine $\langle\langle \mathcal{S}, \mathcal{Q}, \mathcal{F}, \mathcal{T}, q_i\rangle\rangle$ a state q_i is a *terminating* state if there are no transitions of the form $\langle q_i, s_1, s_2, d_3, q_j\rangle$. A Turing Machine is well formed if for every q_i then either q_i is terminating, or for every symbol $s_j \in \mathcal{S}$ then there exists a transition of the form $\langle q_i, s_j, s_1, d_2, q_3\rangle$ for some s_1 and d_2 and q_3. The rest of this paper shall only consider well formed Turing Machines although no results rely upon this.

A Simple Example

Consider the following simple example of a Turing Machine that accepts numbers represented in unary and terminates with the current head on 1 if the number is even and b if the number is odd.

The alphabet is given by $\mathcal{S} = \{b, 1\}$ and the states by $Q = \{q_0, q_1, q_2, q_3\}$. The transition function $\mathcal{F}$ is defined as follows:

$$\begin{array}{lll} \langle q_0, b, 1, L, q_2\rangle & \langle q_1, b, b, L, q_3\rangle & \langle q_2, b, b, R, q_3\rangle \\ \langle q_0, 1, b, R, q_1\rangle & \langle q_1, 1, b, R, q_0\rangle & \langle q_2, 1, b, R, q_3\rangle\,. \end{array}$$

Observe that q_3 is a terminating state and so has not transitions that begin in that state.

Now consider the Turing Machine given by $\langle\langle \mathcal{S}, Q, \mathcal{F}, [\ldots, b, \mathbf{1}, 1, 1, b, \ldots], q_0\rangle\rangle$, that is the Turing Machine defined above with the current head position on the first 1 of the number three represented in unary. The computations progress as follows:

$$\begin{array}{rl} & \langle\langle \mathcal{S}, Q, \mathcal{F}, [\ldots, b, \mathbf{1}, 1, 1, b, \ldots], q_0\rangle\rangle \\ \longmapsto & \langle\langle \mathcal{S}, Q, \mathcal{F}, [\ldots, b, b, \mathbf{1}, 1, b, \ldots], q_1\rangle\rangle \\ \longmapsto & \langle\langle \mathcal{S}, Q, \mathcal{F}, [\ldots, b, b, b, \mathbf{1}, b, \ldots], q_0\rangle\rangle \\ \longmapsto & \langle\langle \mathcal{S}, Q, \mathcal{F}, [\ldots, b, b, b, b, \mathbf{b}, \ldots], q_1\rangle\rangle \\ \longmapsto & \langle\langle \mathcal{S}, Q, \mathcal{F}, [\ldots, b, b, b, \mathbf{b}, b, \ldots], q_3\rangle\rangle\,. \end{array}$$

Since q_3 is a terminating state the Turing Machine halts and has the current head position on a blank as required. A similar Turing Machine with a tape that represents two would have the following reductions

$$\begin{array}{rl} & \langle\langle \mathcal{S}, Q, \mathcal{F}, [\ldots, b, \mathbf{1}, 1, b, \ldots], q_0\rangle\rangle \\ \longmapsto & \langle\langle \mathcal{S}, Q, \mathcal{F}, [\ldots, b, b, \mathbf{1}, b, \ldots], q_1\rangle\rangle \\ \longmapsto & \langle\langle \mathcal{S}, Q, \mathcal{F}, [\ldots, b, b, b, \mathbf{b}, \ldots], q_0\rangle\rangle \\ \longmapsto & \langle\langle \mathcal{S}, Q, \mathcal{F}, [\ldots, b, b, \mathbf{b}, 1, \ldots], q_2\rangle\rangle \\ \longmapsto & \langle\langle \mathcal{S}, Q, \mathcal{F}, [\ldots, b, b, b, \mathbf{1}, \ldots], q_3\rangle\rangle\,. \end{array}$$

Since two is even this time the Turing Machine halts with the current head position on a 1 symbol as required.

3 Intensional Process Calculi

Intensionality in process calculi is the ability for the structure of a term to be determined, here during interaction. This has appeared in communication primitives in some more recent process calculi such as Concurrent Pattern Calculus and Psi Calculi. Spi calculus supports intentional reductions, but not in a communication reduction [1]. This section defines an *asymmetric concurrent pattern calculus* (ACPC) that is a simple variation of Concurrent Pattern Calculus as described before [9]. ACPC is trivial to represent in either Concurrent Pattern Calculus or Psi Calculi, and so has been used here for clarity of the encoding, and to transfer the results (details in Section 5).

Assume a countable collection of names denoted $m, m_1, m_2, n, n_1, \ldots$. The *terms* of ACPC are given by

$$s, t \quad ::= \quad n \mid s \bullet t$$

the *names* n, and *compounds* $s \bullet t$ that combine the two terms s and t into a single term.

The *patterns* of ACPC are defined by

$$p, q \quad ::= \quad \lambda n \mid n \mid p \bullet q\,.$$

The *binding names* λn play the rôle of inputs in the pattern. The *name-match* n is used to test for equality during interaction [1]. The *compound patterns* $p \bullet q$ combine the two patterns p and q into a single pattern. Note that a well-formed pattern is one where each binding name appears only once, the rest of this paper will only consider well formed patterns.

Substitutions, denoted σ, ρ, are finite mappings from names to terms. Their domain and range are expected, with their names being the union of domain and range.

The key to interaction for ACPC is the matching $\{t/\!/p\}$ of the term t against the pattern p to generate a substitution σ is defined as follows.

$$\begin{array}{rcl} \{t/\!/\lambda n\} & \stackrel{\text{def}}{=} & \{t/n\} \\ \{n/\!/n\} & \stackrel{\text{def}}{=} & \{\} \\ \{s \bullet t/\!/p \bullet q\} & \stackrel{\text{def}}{=} & \{s/\!/p\} \cup \{t/\!/q\} \\ \{t/\!/p\} & \text{undefined} & \text{otherwise.} \end{array}$$

Any term t can be matched with a binding name λn to generate a substitution from the binding name to the term $\{t/n\}$. A single name n can be matched with a name-match for that name n to yield the empty substitution. A compound term $s \bullet t$ can be matched by a compound pattern $p \bullet q$ when the components match to yield substitutions $\{s/\!/p\} = \sigma_1$ and $\{t/\!/q\} = \sigma_2$, the resulting substitution is the unification of σ_1 and σ_2. Observe that since patterns are well formed, the substitutions of components will always have disjoint domain. Otherwise the match is undefined.

The processes of ACPC are given by

$$P, Q \quad ::= \quad \mathbf{0} \mid P \mid Q \mid !P \mid (\nu n)P \mid p \to P \mid \overline{t} \to P \,.$$

The null process, parallel composition, replication, and restriction are standard from CPC (and many other process calculi). The *input* $p \to P$ has a pattern p and body P, the binding names of the pattern bind their instances in the body. The *output* $\overline{t} \to P$ has a term t and body P, like in π-calculus and Psi Calculi there are no binding names or scope effects for outputs. Note that an input $p \to \mathbf{0}$ may be denoted by p and an output $\overline{t} \to \mathbf{0}$ by $\overline{t}$ when no ambiguity may occur.

α-conversion $=_\alpha$ is defined upon inputs and restrictions in the usual manner for Concurrent Pattern Calculus [11]. The structural equivalence relation $\equiv$ is given by:

$$\begin{array}{lll} P \mid \mathbf{0} \equiv P & P \mid Q \equiv Q \mid P & P \mid (Q \mid R) \equiv (P \mid Q) \mid R \\ P \equiv P' \quad \text{if } P =_\alpha P' & (\nu a)\mathbf{0} \equiv \mathbf{0} & (\nu a)(\nu b)P \equiv (\nu b)(\nu a)P \\ P \mid (\nu a)Q \equiv (\nu a)(P \mid Q) \quad \text{if } a \notin \mathsf{fn}(P) & & !P \equiv P \mid !P \,. \end{array}$$

The application of a substitution σ to a process P denoted σP is as usual with scope capture avoided by α-conversion where required.

ACPC has a single interaction axiom given by:

$$\overline{t} \to P \mid q \to Q \qquad \longmapsto \qquad P \mid \sigma Q \qquad\qquad \{t/\!/q\} = \sigma \,.$$

It states that when the term of an output can be matched with the pattern of an input to yield the substitution σ then reduce to the body of the output in parallel with σ applied to the body of the input.

[1] This corresponds to the protected names $\ulcorner n \urcorner$ of CPC in the rôle they play. However, the syntax is chosen to mirror the variable names n of CPC since they more closely align with π-calculus and Psi Calculi syntax, and later results for CPC can use either protected or variable names.

4 Encoding

This section presents a faithful encoding of Turing Machines into ACPC. The key to the encoding is to capture the current state and tape as the term of an output, and to encode the transition function as a process that will operate upon the encoded tape. The spirit to this kind of encoding has been captured before when encoding combinatory logics into CPC [9].

Consider the simple encodings $(\!|\cdot|\!)_L$ and $(\!|\cdot|\!)_R$ that take a sequence of symbols and encodes them into a term as follows:

$$\begin{aligned}(\!|\ldots,s_3,s_2,s_1|\!)_L &\stackrel{\text{def}}{=} ((\ldots\bullet s_3)\bullet s_2)\bullet s_1\\ (\!|s_1,s_2,s_3,\ldots|\!)_R &\stackrel{\text{def}}{=} s_1\bullet(s_2\bullet(s_3\bullet\ldots))\,.\end{aligned}$$

That is, $(\!|\cdot|\!)_L$ encodes a sequence of symbols from right to left, compounding on the left hand side. Similarly, $(\!|\cdot|\!)_R$ encodes a sequence of symbols from left to right, compounding on the right hand side.

Now consider a tape that must be of the form $[\ldots,b,s_a,\ldots,s_g,\mathbf{s_h},s_i\ldots,s_j,b,\ldots]$. That is, an infinite sequence of blanks, some sequence of symbols including the current head position, and then another infinite sequence of blanks. This can be encoded $[\![\cdot]\!]$ into a term by:

$$[\![\ [\ldots,b,s_a,\ldots,s_g,\mathbf{s_h},s_i\ldots,s_j,b,\ldots]\ ]\!] \stackrel{\text{def}}{=} ((\!|\ldots,b,s_a,\ldots,s_g|\!)_L)\bullet s_h\bullet((\!|s_i\ldots,s_j,b,\ldots|\!)_R)$$

Observe that the result is a term of the form $a\bullet b\bullet c$ where a is the encoding of the tape left of the current head position, b is the current head position symbol, and c is the encoding of the tape right of the current head position. In particular note that a and c are both compounds of their symbol closest to the current head position and the rest of the sequence in their direction. For now the encoding handles an infinite tape and produces an infinite term, although this can be removed later without effect on the results (details in Section 5).

Now the current state q_i and a tape $\mathcal{T}$ can be represented as an output by

$$\overline{q_i\bullet[\![\mathcal{T}]\!]}\,.$$

Lemma 4.1 *The representation $\overline{q_i\bullet[\![\mathcal{T}]\!]}$ of the state q_i and tape $\mathcal{T}$ does not reduce.*

With the current state and tape encoded into a term it remains to encode the transition function in a manner that allows faithfulness. Consider that each tuple of the transition function is of the form $\langle q_i,s_1,s_2,d,q_j\rangle$ for states q_i and q_j, and symbols s_1 and s_2, and for d either L or R. Thus we can encode $[\![\cdot]\!]$ such a tuple as a single process as follows:

$$\begin{aligned}[\![\langle q_i,s_1,s_2,d,q_j\rangle]\!] &\stackrel{\text{def}}{=} q_i\bullet((\lambda l\bullet\lambda l_1)\bullet s_1\bullet\lambda r)\to\overline{q_j\bullet(l\bullet l_1\bullet(s_2\bullet r))} && d=L\\ [\![\langle q_i,s_1,s_2,d,q_j\rangle]\!] &\stackrel{\text{def}}{=} q_i\bullet(\lambda l\bullet s_1\bullet(\lambda r_1\bullet\lambda r))\to\overline{q_j\bullet((l\bullet s_2)\bullet r_1\bullet r)} && d=R\,.\end{aligned}$$

Note that in both cases the pattern matches on the state q_i and the symbol s_1. When the tape is going to move left then the first symbol to the left is bound to l_1 and the rest to l, and r_1 and r when respectively moving right. The output in each case is the new state q_j and the tape with the written symbol s_2 added to the right side of the head position when moving left, or the left side when moving right. Thus, the new output represents the updated state and tape after the transition $\langle q_i,s_1,s_2,d,q_j\rangle$ has been applied once. Note that the encoding here assumes the four names l and l_1 and r and r_1 do not appear in the symbols $\mathcal{S}$ of the encoded Turing Machine. Since the collection of symbols $\mathcal{S}$ is finite it is always possible to choose four such names.

Building upon this, the encoding of the transition function $\mathcal{F}$ can be done. Define $\prod_{x\in S} P(x)$ to be the parallel composition of processes $P(x)$ in the usual manner. Now the encoding of the transition function $[\![\mathcal{F}]\!]$ can be captured as follows:

$$[\![\mathcal{F}]\!] \stackrel{\text{def}}{=} \prod_{u\in\mathcal{F}} !\,[\![u]\!]$$

where u is each tuple of the form $\langle q_i, s_1, s_2, d, q_j\rangle$. Observe that this creates a process of the form $!P_1 \mid !P_2 \mid \ldots$ where each P_i performs a single transition.

Lemma 4.2 *The encoding $[\![\mathcal{F}]\!]$ of the transition function $\mathcal{F}$ does not reduce.*

Proof For every tuple u in $\mathcal{F}$ the encoding $[\![u]\!]$ is an input. It is then straightforward to consider all the structural congruence rules and show that there are no outputs and thus the reduction axiom cannot be satisfied.

Finally, the encoding $[\![\cdot]\!]$ of a Turing Machine into ACPC is given by:

$$[\![\langle\langle \mathcal{S},\mathcal{Q},\mathcal{F},\mathcal{T},q_i\rangle\rangle]\!] \stackrel{\text{def}}{=} \overline{q_i \bullet [\![\mathcal{T}]\!]} \mid [\![\mathcal{F}]\!]\,.$$

Lemma 4.3 *Given a Turing Machine $\langle\langle \mathcal{S},\mathcal{Q},\mathcal{F},\mathcal{T},q_i\rangle\rangle$ then*

1. *If there is a transition $\langle\langle \mathcal{S},\mathcal{Q},\mathcal{F},\mathcal{T},q_i\rangle\rangle \longmapsto \langle\langle \mathcal{S},\mathcal{Q},\mathcal{F},\mathcal{T}',q_j\rangle\rangle$ then there is a reduction $[\![\langle\langle \mathcal{S},\mathcal{Q},\mathcal{F},\mathcal{T},q_i\rangle\rangle]\!] \longmapsto Q$ where $Q \equiv [\![\langle\langle \mathcal{S},\mathcal{Q},\mathcal{F},\mathcal{T}',q_j\rangle\rangle]\!]$, and*
2. *if there is a reduction $[\![\langle\langle \mathcal{S},\mathcal{Q},\mathcal{F},\mathcal{T},q_i\rangle\rangle]\!] \longmapsto Q$ then $Q \equiv [\![\langle\langle \mathcal{S},\mathcal{Q},\mathcal{F},\mathcal{T}',q_j\rangle\rangle]\!]$ and there is a transition $\langle\langle \mathcal{S},\mathcal{Q},\mathcal{F},\mathcal{T},q_i\rangle\rangle \longmapsto \langle\langle \mathcal{S},\mathcal{Q},\mathcal{F},\mathcal{T}',q_j\rangle\rangle$.*

Proof The first part is proven by examining the tuple u that corresponds to the transition $\langle\langle \mathcal{S},\mathcal{Q},\mathcal{F},\mathcal{T},q_i\rangle\rangle \longmapsto \langle\langle \mathcal{S},\mathcal{Q},\mathcal{F},\mathcal{T}',q_j\rangle\rangle$ by the Turing Machine. This tuple must be of the form $\langle q_i, s_1, s_2, d, q_j\rangle$ and it must also be that $\mathcal{T}$ is of the form $[\ldots, s_m, \mathbf{s_1}, s_n, \ldots]$. Further, it must be that $\mathcal{T}'$ is either: $[\ldots, \mathbf{s_m}, s_2, s_n, \ldots]$ when d is L, or $[\ldots, s_m, s_2, \mathbf{s_n}, \ldots]$ when d is R. Now $[\![\langle\langle \mathcal{S},\mathcal{Q},\mathcal{F},\mathcal{T},q_i\rangle\rangle]\!]$ is of the form $\overline{q_i \bullet [\![\mathcal{T}]\!]} \mid [\![\mathcal{F}]\!]$ and by Lemmas 4.1 and 4.2 neither $\overline{q_i \bullet [\![\mathcal{T}]\!]}$ nor $[\![\mathcal{F}]\!]$ can reduce, respectively. Now by exploiting structural congruence gain that $\overline{q_i \bullet [\![\mathcal{T}]\!]} \mid [\![\mathcal{F}]\!] \equiv \overline{q_i \bullet [\![\mathcal{T}]\!]} \mid [\![\langle q_i, s_1, s_2, d, q_j\rangle]\!] \mid [\![\mathcal{F}]\!]$. Then by the definition of matching and the reduction axiom is straightforward to show that $\overline{q_i \bullet [\![\mathcal{T}]\!]} \mid [\![\langle q_i, s_1, s_2, d, q_j\rangle]\!] \longmapsto \overline{q_j \bullet [\![\mathcal{T}']\!]}$ and thus conclude.

The reverse direction is proved similarly by observing that the only possible reduction $[\![\langle\langle \mathcal{S},\mathcal{Q},\mathcal{F},\mathcal{T},q_i\rangle\rangle]\!] \longmapsto Q$ must be due to a tuple $\langle q_i, s_1, s_2, d, q_j\rangle$ that is in the transition function $\mathcal{F}$ and the result follows.

Theorem 4.4 *The encoding $[\![\cdot]\!]$ of a Turing Machine into ACPC; faithfully preserves reduction, and divergence. That is, given a Turing Machine $\langle\langle \mathcal{S},\mathcal{Q},\mathcal{F},\mathcal{T},q_i\rangle\rangle$ then it holds that:*

1. *there is a transition $\langle\langle \mathcal{S},\mathcal{Q},\mathcal{F},\mathcal{T},q_i\rangle\rangle \longmapsto \langle\langle \mathcal{S},\mathcal{Q},\mathcal{F},\mathcal{T}',q_j\rangle\rangle$ if and only if there is exactly one reduction $[\![\langle\langle \mathcal{S},\mathcal{Q},\mathcal{F},\mathcal{T},q_i\rangle\rangle]\!] \longmapsto Q$ where $Q \equiv [\![\langle\langle \mathcal{S},\mathcal{Q},\mathcal{F},\mathcal{T}',q_j\rangle\rangle]\!]$, and*
2. *there is an infinite sequence of transitions $\langle\langle \mathcal{S},\mathcal{Q},\mathcal{F},\mathcal{T},q_i\rangle\rangle \longmapsto^{\omega}$ if and only if there is an infinite sequence of reductions $[\![\langle\langle \mathcal{S},\mathcal{Q},\mathcal{F},\mathcal{T},q_i\rangle\rangle]\!] \longmapsto^{\omega}$.*

Proof Both parts can be proved by exploiting Lemma 4.3.

Observe that this encoding of a Turing Machine into ACPC is not only faithful and straightforward, but also up to structural congruence. This is in contrast with the popular style of encoding λ-calculi into process calculi that requires many reductions to simulate one λ-reduction, and the equivalence of encoded terms/machines is only up to weak behavioural equivalence. The simplicity and faithfulness here is gained by being able to directly render the current state and tape as a single term, and the transition function as a process that modifies the current state and tape in the same manner as the original Turing Machine.

5 Variations

This section considers variations to the encoding including: representing the tape as a finite term, encoding into Concurrent Pattern Calculus, and encoding into Psi Calculi.

Finite Terms

One potential concern is the infinite tape being represented as an infinite term in ACPC. However, this can be done away with by adding an additional reserved name e during the translation that does not appear in the symbols of the Turing Machine $\mathcal{S}$ and represents the edge of the tape.

Now $(\!|\cdot|\!)_L$ and $(\!|\cdot|\!)_R$ are modified to account for the endless sequence of blank symbols b as follows:

$$\begin{aligned}(\!|\ldots, b, s_i, \ldots, s_1|\!)_L &\stackrel{\text{def}}{=} ((e \bullet s_i)\ldots) \bullet s_1\\ (\!|s_1, \ldots, s_i, b, \ldots|\!)_R &\stackrel{\text{def}}{=} s_1 \bullet (\ldots(s_i \bullet e))\,.\end{aligned}$$

Here the endless blanks at the edge of the tape are simply replaced by e. Otherwise the encoding of the state q_i and tape $\mathcal{T}$ is the same.

Lemma 5.1 *The representation $\overline{q_i \bullet [\![\mathcal{T}]\!]}$ of the state q_i and tape $\mathcal{T}$ does not reduce.*

The encoding of tuples $[\![\cdot]\!]$ is now modified to account for e given by:

$$\begin{aligned}[\![\langle q_i, s_1, s_2, d, q_j\rangle]\!] &\stackrel{\text{def}}{=} !q_i \bullet ((\lambda l \bullet \lambda l_1) \bullet s_1 \bullet \lambda r) \to \overline{q_j \bullet (l \bullet l_1 \bullet (s_2 \bullet r))}\\ &\quad\ \mid !q_i \bullet (e \bullet s_1 \bullet \lambda r) \to \overline{q_j \bullet (e \bullet b \bullet (s_2 \bullet r))} \qquad d = L\\ [\![\langle q_i, s_1, s_2, d, q_j\rangle]\!] &\stackrel{\text{def}}{=} !q_i \bullet (\lambda l \bullet s_1 \bullet (\lambda r_1 \bullet \lambda r)) \to \overline{q_j \bullet ((l \bullet s_2) \bullet r_1 \bullet r)}\\ &\quad\ \mid !q_i \bullet (\lambda l \bullet s_1 \bullet e) \to \overline{q_j \bullet ((l \bullet s_2) \bullet b \bullet e)} \qquad d = R\,.\end{aligned}$$

The encoding of a tuple now has two input processes in parallel and each under a replication; the first matching the original encoding, and the second detecting when the transition would move the current head position to the edge of the tape. The new one inserts a new blank b in the output and shifts the edge e along one cell. Observe that due to definition of the matching rule no output can interact with both of these inputs (as no term can be matched with both patterns of the form $\lambda m \bullet \lambda n$ and e). The replications have been added so that structural congruence can be achieved in the final results. This requires a change to the encoding of the transitions function as follows:

$$[\![\mathcal{F}]\!] \stackrel{\text{def}}{=} \prod_{u \in \mathcal{F}} [\![u]\!]$$

where the replications are now left to the encoding of each tuple $[\![u]\!]$.

The rest of the results follow with minor alterations.

Lemma 5.2 *The encoding* $[\![\mathcal{F}]\!]$ *of the transition function* $\mathcal{F}$ *does not reduce.*

Lemma 5.3 *Given a Turing Machine* $\langle\langle \mathcal{S},\mathcal{Q},\mathcal{F},\mathcal{T},q_i\rangle\rangle$ *then*

1. *If there is a transition* $\langle\langle \mathcal{S},\mathcal{Q},\mathcal{F},\mathcal{T},q_i\rangle\rangle \longmapsto \langle\langle \mathcal{S},\mathcal{Q},\mathcal{F},\mathcal{T}',q_j\rangle\rangle$ *then there is a reduction* $[\![\langle\langle \mathcal{S},\mathcal{Q},\mathcal{F},\mathcal{T},q_i\rangle\rangle]\!] \longmapsto Q$ *where* $Q \equiv [\![\langle\langle \mathcal{S},\mathcal{Q},\mathcal{F},\mathcal{T}',q_j\rangle\rangle]\!]$*, and*
2. *if there is a reduction* $[\![\langle\langle \mathcal{S},\mathcal{Q},\mathcal{F},\mathcal{T},q_i\rangle\rangle]\!] \longmapsto Q$ *then* $Q \equiv [\![\langle\langle \mathcal{S},\mathcal{Q},\mathcal{F},\mathcal{T}',q_j\rangle\rangle]\!]$ *and there is a transition* $\langle\langle \mathcal{S},\mathcal{Q},\mathcal{F},\mathcal{T},q_i\rangle\rangle \longmapsto \langle\langle \mathcal{S},\mathcal{Q},\mathcal{F},\mathcal{T}',q_j\rangle\rangle$.

Proof The first part is proven by examining the tuple u that corresponds to the transition $\langle\langle \mathcal{S},\mathcal{Q},\mathcal{F},\mathcal{T},q_i\rangle\rangle \longmapsto \langle\langle \mathcal{S},\mathcal{Q},\mathcal{F},\mathcal{T}',q_j\rangle\rangle$ by the Turing Machine. This tuple must be of the form $\langle q_i,s_1,s_2,d,q_j\rangle$ and it must also be that $\mathcal{T}$ is of the form $[\ldots,s_m,\mathbf{s_1},s_n,\ldots]$. Further, it must be that $\mathcal{T}'$ is either: $[\ldots,\mathbf{s_m},s_2,s_n,\ldots]$ when d is L, or $[\ldots,s_m,s_2,\mathbf{s_n},\ldots]$ when d is R. Now $[\![\langle\langle \mathcal{S},\mathcal{Q},\mathcal{F},\mathcal{T},q_i\rangle\rangle]\!]$ is of the form $\overline{q_i \bullet [\![\mathcal{T}]\!]} \mid [\![\mathcal{F}]\!]$ and by Lemmas 5.1 and 5.2 neither $q_i \bullet [\![\mathcal{T}]\!]$ nor $[\![\mathcal{F}]\!]$ can reduce, respectively.

By definition $[\![\mathcal{F}]\!]$ is of the form $[\![u]\!] \mid R$ for some process R. Now consider d.

- If d is L then consider the encoded tape $[\![\mathcal{T}]\!]$.
  - If $[\![\mathcal{T}]\!]$ is of the form $e \bullet s_1 \bullet t$ then by definition of $[\![u]\!]$ and structural congruence $[\![u]\!] \equiv q_i \bullet (e \bullet s_1 \bullet \lambda r) \to \overline{q_j \bullet (e \bullet b \bullet (s_2 \bullet r))} \mid [\![u]\!]$ and thus there is a reduction $\overline{q_i \bullet [\![\mathcal{T}]\!]} \mid q_i \bullet (e \bullet s_1 \bullet \lambda r) \to \overline{q_j \bullet (e \bullet b \bullet (s_2 \bullet r))} \mid [\![u]\!] \longmapsto \overline{q_j \bullet (e \bullet b \bullet (s_2 \bullet r))} \mid [\![u]\!]$. It is straightforward to show that $[\![\mathcal{T}']\!] = e \bullet b \bullet (s_2 \bullet r)$ and thus by structural congruence that $\overline{q_j \bullet (e \bullet b \bullet (s_2 \bullet r))} \mid [\![u]\!] \mid R \equiv \overline{q_j \bullet [\![\mathcal{T}']\!]} \mid [\![\mathcal{F}]\!]$ and thus conclude.
  - If $[\![\mathcal{T}]\!]$ is of the form $(s \bullet s_i) \bullet s_1 \bullet r$ then take $[\![u]\!] \equiv q_i \bullet ((\lambda l \bullet \lambda l_1) \bullet s_1 \bullet \lambda r) \to \overline{q_j \bullet (l \bullet l_1 \bullet (s_2 \bullet r))} \mid [\![u]\!]$ and the rest is as in the previous case.
- If d is R then the proof is a straightforward adaptation of the L case above.

The reverse direction is proved similarly by observing that the only possible reduction $[\![\langle\langle \mathcal{S},\mathcal{Q},\mathcal{F},\mathcal{T},q_i\rangle\rangle]\!] \longmapsto Q$ must be due to a tuple $\langle q_i,s_1,s_2,d,q_j\rangle$ that is in the transition function $\mathcal{F}$ and the result follows. The only added complexity is to ensure that there is only one possible reduction for a given current state and current head position symbol, this can be assured by definition of the match rule excluding any term from matching with both patterns $\lambda l \bullet \lambda l_1$ and e.

Theorem 5.4 *The encoding* $[\![\cdot]\!]$ *of a Turing Machine into ACPC; faithfully preserves reduction, and divergence. That is, given a Turing Machine* $\langle\langle \mathcal{S},\mathcal{Q},\mathcal{F},\mathcal{T},q_i\rangle\rangle$ *then it holds that:*

1. *there is a transition* $\langle\langle \mathcal{S},\mathcal{Q},\mathcal{F},\mathcal{T},q_i\rangle\rangle \longmapsto \langle\langle \mathcal{S},\mathcal{Q},\mathcal{F},\mathcal{T}',q_j\rangle\rangle$ *if and only if there is exactly one reduction* $[\![\langle\langle \mathcal{S},\mathcal{Q},\mathcal{F},\mathcal{T},q_i\rangle\rangle]\!] \longmapsto Q$ *where* $Q \equiv [\![\langle\langle \mathcal{S},\mathcal{Q},\mathcal{F},\mathcal{T}',q_j\rangle\rangle]\!]$*, and*
2. *there is an infinite sequence of transitions* $\langle\langle \mathcal{S},\mathcal{Q},\mathcal{F},\mathcal{T},q_i\rangle\rangle \longmapsto^{\omega}$ *if and only if there is an infinite sequence of reductions* $[\![\langle\langle \mathcal{S},\mathcal{Q},\mathcal{F},\mathcal{T},q_i\rangle\rangle]\!] \longmapsto^{\omega}$.

Proof Both parts can be proved by exploiting Lemma 5.3.

Concurrent Pattern Calculus

The choice of using ACPC here rather than CPC is for simplicity in presentation. This section recalls CPC and proves that the encodings hold in CPC as well. CPC has a single class of *patterns* that combines both the terms and patterns of ACPC defined as follows:

$$p,q \quad ::= \quad \lambda x \mid n \mid \ulcorner n \urcorner \mid p \bullet q \,.$$

The *binding names* λx are as before. The *variable names* n can be used as both output (like the name terms of ACPC) and equality tests (like the name-match of ACPC). The *protected names* $\ulcorner n \urcorner$ are only equality tests (name-matches of ACPC). *Compounds* $p \bullet q$ are as in ACPC. A *communicable pattern* is a pattern that contains no binding or protected names.

Interaction CPC relies upon the *unification* $\{p \parallel q\}$ of the patterns p and q to yield a pair of substitutions (σ,ρ) and is defined by:

$$\left.\begin{array}{r} \{x \parallel x\} \\ \{x \parallel \ulcorner x \urcorner\} \\ \{\ulcorner x \urcorner \parallel x\} \\ \{\ulcorner x \urcorner \parallel \ulcorner x \urcorner\} \end{array}\right\} \stackrel{\text{def}}{=} (\{\},\{\})$$

$$\{\lambda x \parallel q\} \stackrel{\text{def}}{=} (\{q/x\},\{\}) \qquad q \text{ is communicable}$$

$$\{p \parallel \lambda x\} \stackrel{\text{def}}{=} (\{\},\{p/x\}) \qquad p \text{ is communicable}$$

$$\{p_1 \bullet p_2 \parallel q_1 \bullet q_2\} \stackrel{\text{def}}{=} (\sigma_1 \cup \sigma_2, \rho_1 \cup \rho_2) \quad \{p_i \parallel q_i\} = (\sigma_i,\rho_i)\ i \in \{1,2\}$$

$$\{p \parallel q\} \quad \text{undefined} \quad \text{otherwise.}$$

The unification succeeds and yields empty substitutions when both patterns are the same name and are both variable or protected. If either pattern is a binding name and the other is communicable, then the communicable pattern is bound to the binding name in the appropriate substitution. Otherwise if both patterns are compounds then unify component-wise. Finally, if all these fail then unification is undefined (impossible).

The process of CPC are given by:

$$P,Q \quad ::= \quad \mathbf{0} \mid P|Q \mid !P \mid (\nu n)P \mid p \to P\,.$$

All are familiar from ACPC although the input and output are now both represented by the *case* $p \to P$ with pattern p and body P.

The structural laws are the same as for ACPC with α-conversion defined in the usual manner [11, 12] and interaction is defined by the following axiom:

$$p \to P \mid q \to Q \quad \longmapsto \quad (\sigma P) \mid (\rho Q) \qquad \{p \parallel q\} = (\sigma,\rho)\,.$$

It states that when two cases in parallel can unify their patterns to yield substitutions σ and ρ then apply those substitutions to the appropriate bodies.

The encodings of Turing Machines into CPC are trivial, the only change is to remove the overhead line from outputs, i.e. $\overline{t} \to P$ becomes $t \to P$ since all terms of ACPC are patterns of CPC [2]. However some proofs require changes due to the change from input and output with one sided matching, to CPC cases with pattern unification. The proof of Lemma 4.1 is trivial. For Lemmas 4.2 and 5.2 the proof is resolved due to CPC unification only allowing a binding name λx to unify with a communicable pattern. The rest are effectively unchanged.

Theorem 5.5 *There is an encoding* $[\![\cdot]\!]$ *of a Turing Machine into CPC that; faithfully preserves reduction, and divergence. That is, given a Turing Machine* $\langle\langle \mathcal{S},\mathcal{Q},\mathcal{F},\mathcal{T},q_i\rangle\rangle$ *then it holds that:*

[2] There is no need to convert syntax between ACPC and CPC, for example changing name-matches from n to $\ulcorner n \urcorner$ in patterns, as the unification rules for CPC allow for both. Indeed, the encodings were chosen to allow this. Although in theory CPC could allow two ACPC outputs to interact, this does not occur for the encodings in this paper.

1. *there is a transition* $\langle\langle \mathcal{S},\mathcal{Q},\mathcal{F},\mathcal{T},q_i\rangle\rangle \longmapsto \langle\langle \mathcal{S},\mathcal{Q},\mathcal{F},\mathcal{T}',q_j\rangle\rangle$ *if and only if there is exactly one reduction* $[\![\langle\langle \mathcal{S},\mathcal{Q},\mathcal{F},\mathcal{T},q_i\rangle\rangle]\!] \longmapsto Q$ *where* $Q \equiv [\![\langle\langle \mathcal{S},\mathcal{Q},\mathcal{F},\mathcal{T}',q_j\rangle\rangle]\!]$*, and*
2. *there is an infinite sequence of transitions* $\langle\langle \mathcal{S},\mathcal{Q},\mathcal{F},\mathcal{T},q_i\rangle\rangle \longmapsto^{\omega}$ *if and only if there is an infinite sequence of reductions* $[\![\langle\langle \mathcal{S},\mathcal{Q},\mathcal{F},\mathcal{T},q_i\rangle\rangle]\!] \longmapsto^{\omega}$.

Psi Calculi

Similarly both encodings can be easily adapted for Psi Calculi [3]. Psi Calculi are parametrized with respect to two sets: terms $\mathbf{T}$ defined by

$$M,N \quad ::= \quad m \mid M,N$$

the names m and the *pair* M,N of two terms M and N; and assertions $\mathbf{A}$, ranged over by Ψ (that play no significant rôle here). The empty assertion is written $\mathbf{1}$. Also assume two operators: channel equivalence, $\dot{\leftrightarrow} \subseteq \mathbf{T}\times\mathbf{T}$, and assertion composition, $\otimes : \mathbf{A}\times\mathbf{A}\to\mathbf{A}$. It is also required that $\dot{\leftrightarrow}$ is transitive and symmetric, and that $(\otimes,\mathbf{1})$ is a commutative monoid.

Processes in Psi Calculi are defined as:

$$P,Q \ ::= \ \mathbf{0} \mid P|Q \mid (\nu x)P \mid !P \mid \underline{M}(\lambda\widetilde{x})N.P \mid \overline{M}(N).P \mid (\!|\Psi|\!)$$

exploiting the notation $\widetilde{a}$ for a sequence $a_1,\ldots,a_i$. Most process forms are as usual with: *input* $\underline{M}(\lambda\widetilde{x})N.P$ on channel M and binding names $\widetilde{x}$ in the pattern N and with body P; and *output* $\overline{M}(N).P$ on channel M and outputting term N.

The reduction relation semantics are given by isolating the τ actions of the LTS given in [3]. To this aim, the definiton of frame of a process P, written $\mathcal{F}(P)$, as the set of unguarded assertions occurring in P. Formally:

$$\mathcal{F}((\!|\Psi|\!)) = \Psi \qquad \mathcal{F}((\nu x)P) = (\nu x)\mathcal{F}(P) \qquad \mathcal{F}(P|Q) = \mathcal{F}(P)\otimes\mathcal{F}(Q)$$

and is $\mathbf{1}$ in all other cases. Denote as $(\nu\widetilde{b_P})\Psi_P$ the frame of P. The structural laws are the same as in ACPC. The reduction relation is inferred by the following axioms:

$$\frac{\Psi \vdash M \dot{\leftrightarrow} N}{\Psi \triangleright \overline{M}(K).P \mid \underline{N}(\lambda\widetilde{x})H.Q \longmapsto P \mid \{\widetilde{L}/\widetilde{x}\}Q}\ K = H[\widetilde{x} := \widetilde{L}] \qquad \frac{\Psi \triangleright P \longmapsto P'}{\Psi \triangleright (\nu x)P \longmapsto (\nu x)P'}\ x \notin \text{names}(\Psi)$$

$$\frac{\Psi\otimes\Psi_Q \triangleright P \longmapsto P'}{\Psi \triangleright P \mid Q \longmapsto P' \mid Q}\ \mathcal{F}(Q) = (\nu\widetilde{b_Q})\Psi_Q, \widetilde{b_Q} \text{ fresh for } \Psi \text{ and } P \qquad \frac{P \equiv Q \quad \Psi \triangleright Q \longmapsto Q' \quad Q' \equiv P'}{\Psi \triangleright P \longmapsto P'}$$

The interesting axiom is the first that states when M and N are equivalent and the term K is equal to the term H with each name in $x_i \in \widetilde{x}$ replaced by some L_i, then reduce to P in parallel with the substitution $\{L_i/x_i\}$ applied to Q. Denote with $P \longmapsto P'$ whenever $\mathbf{1} \triangleright P \longmapsto P'$.

Both encodings of Turing Machines into ACPC can be easily adapted for Psi Calculi, the following changes show how to do adapt the encodings for the infinite tape encoding. All instances of the compounding operator $\bullet$ are replaced by the pair operator ",", i.e. all terms and patterns of the form $x \bullet y$ take the form x,y. The encoding of the current state q_i and tape $\mathcal{T}$ becomes:

$$\overline{q_i}(\ [\![\mathcal{T}]\!]\ ).\mathbf{0}\ .$$

The encoding of the tuples becomes:

$$[\![\langle q_i, s_1, s_2, d, q_j\rangle]\!] \stackrel{\text{def}}{=} \underline{q_i}(\lambda l_1, \lambda l, \lambda r)((l, l_1), s_1, r) \,.\, \overline{q_j}(l, l_1, (s_2, r)).\mathbf{0} \qquad d = L$$
$$[\![\langle q_i, s_1, s_2, d, q_j\rangle]\!] \stackrel{\text{def}}{=} \underline{q_i}(\lambda l, \lambda r_1, \lambda r)(l, s_1, (r_1, r)) \,.\, \overline{q_j}((s_2, l), r_1, r).\mathbf{0} \qquad d = R\,.$$

From there the rest of the encoding remains the same and the results are straightforward.

Theorem 5.6 *The encoding $[\![\cdot]\!]$ of a Turing Machine into Psi Calculi; faithfully preserves reduction, and divergence. That is, given a Turing Machine $\langle\langle \mathcal{S}, \mathcal{Q}, \mathcal{F}, \mathcal{T}, q_i\rangle\rangle$ then it holds that:*

1. *there is a transition $\langle\langle \mathcal{S}, \mathcal{Q}, \mathcal{F}, \mathcal{T}, q_i\rangle\rangle \longmapsto \langle\langle \mathcal{S}, \mathcal{Q}, \mathcal{F}, \mathcal{T}', q_j\rangle\rangle$ if and only if there is exactly one reduction $[\![\langle\langle \mathcal{S}, \mathcal{Q}, \mathcal{F}, \mathcal{T}, q_i\rangle\rangle]\!] \longmapsto Q$ where $Q \equiv [\![\langle\langle \mathcal{S}, \mathcal{Q}, \mathcal{F}, \mathcal{T}', q_j\rangle\rangle]\!]$, and*
2. *there is an infinite sequence of transitions $\langle\langle \mathcal{S}, \mathcal{Q}, \mathcal{F}, \mathcal{T}, q_i\rangle\rangle \longmapsto^{\omega}$ if and only if there is an infinite sequence of reductions $[\![\langle\langle \mathcal{S}, \mathcal{Q}, \mathcal{F}, \mathcal{T}, q_i\rangle\rangle]\!] \longmapsto^{\omega}$.*

6 Limitations

This section discusses the difficulties of attempting to faithfully encode Turing Machines into non-intensional calculi, particularly π-calculi. (Here for synchronous polyadic π-calculus, but adaptations for asynchronous or monadic variations are straightforward, although may require more reductions.)

The π-calculus processes are given by the following grammar:

$$P ::= \mathbf{0} \mid P \mid Q \mid !P \mid (\nu n)P \mid a(\widetilde{x}).P \mid \overline{a}\langle\widetilde{b}\rangle.P\,.$$

The null process, parallel composition, replication, and restriction are as usual. The input $a(\widetilde{x}).P$ has channel name a and a sequence of binding names $x_1, x_2, \ldots, x_i$ denoted by $\widetilde{x}$ and body P. The output $\overline{a}\langle\widetilde{b}\rangle.P$ has channel name a and sequence of output names $b_1, b_2, \ldots, b_i$ denoted $\widetilde{b}$ and body P. The length of a sequence $\widetilde{x}$ is denoted $|\widetilde{x}|$, i.e. $|x_1, x_2, \ldots, x_k| = k$. α-conversion and structural equivalence are as usual.

The only reduction axiom is

$$\overline{m}\langle\widetilde{b}\rangle.P \mid n(\widetilde{x}).Q \quad \longmapsto \quad P \mid \{b_i/x_i\}Q \qquad\qquad m = n \text{ and } |\widetilde{b}| = |\widetilde{x}|\,.$$

That is an output and input reduce if they have the same channel name and the length of their output names and binding names are the same, reducing to the body of the output, in parallel with the substitution that binds each output name b_i to the corresponding binding name x_i applied to the body of the input. The reduction relation is obtained by closing this reduction rule by parallel, restriction and the same structural congruence relation defined for ACPC.

The first limitation of π-calculi is in the number of names that can be determined equal in an interaction. In the Psi Calculi encoding the channel name is used to detect the state, this can also be used for π-calculi as well. However, the detection of the symbol at the current head position would require an additional reduction. There are two approaches that can resolve this first limitation while maintaining faithfulness. The first solution is to account for both names by representing every possible pair of state and symbol by a new name. That is, the encoding of a transition tuple can be represented by

$$[\![\langle q_1, s_2, s_3, d_4, q_5\rangle]\!] \stackrel{\text{def}}{=} q_1 s_2(\ldots).P$$

for some form of input $(\ldots)$ and process P. Here the state q_1 and current head symbol s_2 are combined into a single name $q_1 s_2$ by the encoding. The second approach is to use a structural equivalence rule such as $\mathsf{if}\ m = n\ \mathsf{then}\ P\ \mathsf{else}\ Q$ with the following rules

$$\mathsf{if}\ m = m\ \mathsf{then}\ P\ \mathsf{else}\ Q \equiv P \qquad\qquad \mathsf{if}\ m = n\ \mathsf{then}\ P\ \mathsf{else}\ Q \equiv Q \quad m \neq n\ .$$

Now the encoding of all of the tuples of the form $\langle q_1, s_a, s_b, d, q_c\rangle$ are of the form

$$[\![\, \langle q_1, s_a, s_b, d, q_c\rangle \,]\!] \stackrel{\text{def}}{=} q_1(x,\ldots).\mathsf{if}\ x = s_1\ \mathsf{then}\ P1\ \mathsf{else}\ \mid \ldots \mid \mathsf{if}\ x = s_i\ \mathsf{then}\ P_i\ \mathsf{else}\ \mathbf{0}$$

for each possible symbol $s_1, s_2, \ldots, s_i \in \mathcal{S}$. Here P_j represents the process that does the transition for $\langle q_1, s_j, s_x, d, q_y\rangle$, that is the reductions that correspond to the transition for the matching current head position symbol s_j, and are $\mathbf{0}$ otherwise.

Note that there are other solutions to the problem of matching the state, such as doing further reductions after binding the symbol at the current head position, however these would immediately fail faithfulness.

The impossibility of encoding Turing Machines faithfully without intensionality arises from the encoding of the tape into a single structure. Since π-calculi cannot bind a structured term to a single name, it is impossible to represent the infinite tape by a finite structure. The closest is to take the traditional approach of using some name(s) to identify where the rest of the structure (tape) can be obtained from. For example, consider the followings of an encoding into π-calculus:

$$\begin{aligned}
(\!|\ldots, b, s_i, \ldots, s_1|\!)_L &\stackrel{\text{def}}{=} (\nu x_1)\overline{l}\langle s_1, x_1\rangle \mid (\nu x_2)\overline{x_1}\langle s_2, x_2\rangle \mid \ldots (\nu x_i)\overline{x_{i-1}}\langle s_i, x_i\rangle \mid (\nu x_{i+1})\overline{x_i}\langle b, x_{i+1}\rangle \mid \ldots \\
(\!|s_1, \ldots, s_i, b, \ldots|\!)_R &\stackrel{\text{def}}{=} (\nu x_1)\overline{r}\langle s_1, x_1\rangle \mid (\nu x_2)\overline{x_1}\langle s_2, x_2\rangle \mid \ldots (\nu x_i)\overline{x_{i-1}}\langle s_i, x_i\rangle \mid (\nu x_{i+1})\overline{x_i}\langle b, x_{i+1}\rangle \mid \ldots\ .
\end{aligned}$$

where l and r are reserved names for the left and right hand sides of the tape, respectively, in the encoding. Observe that in each case, the name can be used as a channel to input the symbol to the left l or right r and the next name to use for the next symbol in that direction. Note that parallel composition is used as this allows results to exploit structural equivalence.

Using this approach the state q_i and tape $[\ldots, b, s_a, \ldots, s_g, \mathbf{s_h}, s_i \ldots, s_j, b, \ldots]$ can be encoded by

$$[\![\ [\ldots, b, s_a, \ldots, s_g, \mathbf{s_h}, s_i \ldots, s_j, b, \ldots]\ ]\!]_{q_i} \stackrel{\text{def}}{=} q_i s_h\langle l, r\rangle.\mathbf{0} \mid (\!|\ldots, b, s_a, \ldots, s_g|\!)_L \mid (\!|s_i \ldots, s_j, b, \ldots|\!)_R\ .$$

Now a transition $\langle q_i, s_1, s_2, d, q_j\rangle$ can be encoded as follows (showing the $d = L$ case only):

$$\begin{aligned}
[\![\, \langle q_i, s_1, s_2, L, q_j\rangle \,]\!] \stackrel{\text{def}}{=}\ & q_i s_1(l_c, r_c). \\
& l_c(s_c, l_1). \\
& (\nu r_1) \\
& (\mathsf{if}\ s_c = s_{00}\ \mathsf{then}\ q_j s_{00}\langle l_1, r_1\rangle\ \mathsf{else}\ \ldots \mathsf{if}\ s_c = s_{kk}\ \mathsf{then}\ q_j s_{kk}\langle l_1, r_1\rangle\ \mathsf{else}\ \mathbf{0} \\
& \mid r_1 < s_2, r_c >)
\end{aligned}$$

where each line after the encoding does as follows. The $q_i s_1(l_c, r_c).$ matches the (encoded) state q_i and current head position symbol s_1, and binds the names to access the left and right parts of the tape to l_c and r_c, respectively. Since the transition moves left, the $l_c(s_c, l_1).$ then reads the next symbol to the left s_c and the name to access the rest of the left hand side of the tape l_1. A new name for the new right hand side of the tape is created with (νr_1). The next line detects the new symbol s_c under the current head

position by comparing to each possible symbol $s_{00}, s_{01}, \ldots, s_{kk} \in \mathcal{S}$ and outputting the new left and right hand tape access channel names l_1 and r_1 respectively on the appropriately encoded channel name $q_j s_c$. Finally, in parallel $r_1\langle s_2, r_c\rangle$ provides the new right hand side of the tape. The encoding of the $d = R$ transitions can be done similarly.

Putting all of these pieces together as in Section 4 allows similar results to ACPC to be applied to π-calculi.

Lemma 6.1 *The representation* $[\![\ [\ldots, b, s_a, \ldots, s_g, \mathbf{s_h}, s_i \ldots, s_j, b, \ldots]\ ]\!]_{q_i} \stackrel{\text{def}}{=} q_i s_h\langle l, r\rangle.\mathbf{0} \mid (\!|\ldots, b, s_a, \ldots, s_g|\!)_L \mid (\!|s_i \ldots, s_j, b, \ldots|\!)_R$. *of the state* q_i *and tape* $[\ldots, b, s_a, \ldots, s_g, \mathbf{s_h}, s_i \ldots, s_j, b, \ldots]$ *does not reduce.*

Lemma 6.2 *The encoding* $[\![\mathcal{F}]\!] \stackrel{\text{def}}{=} \prod_{u \in \mathcal{F}} ![\![u]\!]$ *where u is each tuple of the form* $\langle q_i, s_1, s_2, d, q_j\rangle$ *of the transition function* $\mathcal{F}$ *does not reduce.*

Finally, the encoding $[\![\cdot]\!]$ of a Turing Machine into π-calculus is given by:

$$[\![\langle\langle \mathcal{S}, \mathcal{Q}, \mathcal{F}, \mathcal{T}, q_i\rangle\rangle]\!] \quad \stackrel{\text{def}}{=} \quad (\nu l)(\nu r)(\ [\![\mathcal{T}]\!]_{q_i} \mid [\![\mathcal{F}]\!]\ ) .$$

The limitations of π-calculi appear in the following lemma where the correspondence of a single reduction in the original to a single reduction in the translation is lost, instead a single reduction becomes two reductions (or more for some other π-calculi). Further, the proof is complicated by now having to consider α-equivalence of all the restricted names. The impact is also in the final theorem in this section where faithfulness is lost.

Lemma 6.3 *Given a Turing Machine* $\langle\langle \mathcal{S}, \mathcal{Q}, \mathcal{F}, \mathcal{T}, q_i\rangle\rangle$ *then*

1. *If there is a transition* $\langle\langle \mathcal{S}, \mathcal{Q}, \mathcal{F}, \mathcal{T}, q_i\rangle\rangle \longmapsto \langle\langle \mathcal{S}, \mathcal{Q}, \mathcal{F}, \mathcal{T}', q_j\rangle\rangle$ *then there are reductions* $[\![\langle\langle \mathcal{S}, \mathcal{Q}, \mathcal{F}, \mathcal{T}, q_i\rangle\rangle]\!] \longmapsto\!\!\!\!\longmapsto Q$ *where* $Q \equiv [\![\langle\langle \mathcal{S}, \mathcal{Q}, \mathcal{F}, \mathcal{T}', q_j\rangle\rangle]\!]$, *and*
2. *if there is a reduction* $[\![\langle\langle \mathcal{S}, \mathcal{Q}, \mathcal{F}, \mathcal{T}, q_i\rangle\rangle]\!] \longmapsto Q$ *then there exists* Q' *such that* $Q \longmapsto Q'$ *and* $Q' \equiv [\![\langle\langle \mathcal{S}, \mathcal{Q}, \mathcal{F}, \mathcal{T}', q_j\rangle\rangle]\!]$ *and there is a transition* $\langle\langle \mathcal{S}, \mathcal{Q}, \mathcal{F}, \mathcal{T}, q_i\rangle\rangle \longmapsto \langle\langle \mathcal{S}, \mathcal{Q}, \mathcal{F}, \mathcal{T}', q_j\rangle\rangle$.

Proof The first part is proven by examining the tuple u that corresponds to the transition $\langle\langle \mathcal{S}, \mathcal{Q}, \mathcal{F}, \mathcal{T}, q_i\rangle\rangle \longmapsto \langle\langle \mathcal{S}, \mathcal{Q}, \mathcal{F}, \mathcal{T}', q_j\rangle\rangle$ by the Turing Machine. This tuple must be of the form $\langle q_i, s_1, s_2, d, q_j\rangle$ and it must also be that $\mathcal{T}$ is of the form $[\ldots, s_k, s_m, \mathbf{s_1}, s_n, s_o, \ldots]$. Further, it must be that $\mathcal{T}'$ is either: $[\ldots, s_k, \mathbf{s_m}, s_2, s_n, s_o, \ldots]$ when d is L, or $[\ldots, s_k, s_m, s_2, \mathbf{s_n}, s_o, \ldots]$ when d is R. Now $[\![\langle\langle \mathcal{S}, \mathcal{Q}, \mathcal{F}, \mathcal{T}, q_i\rangle\rangle]\!]$ is of the form $(\nu l)(\nu r)(\ [\![\mathcal{T}]\!]_{q_i} \mid [\![\mathcal{F}]\!]\ )$ and by Lemmas 6.1 and 6.2 neither $[\![\mathcal{T}]\!]_{q_i}$ nor $[\![\mathcal{F}]\!]$ can reduce, respectively. Now by exploiting structural congruence gain that $(\nu l)(\nu r)(\ [\![\mathcal{T}]\!]_{q_i} \mid [\![\mathcal{F}]\!]\ ) \equiv (\nu l)(\nu r)(\ [\![\mathcal{T}]\!]_{q_i} \mid [\![\langle q_i, s_1, s_2, d, q_j\rangle]\!] \mid [\![\mathcal{F}]\!]\ )$. Then by two applications of the definition of the π-calculus reduction axiom it is straightforward to show that $[\![\mathcal{T}]\!]_{q_i} \mid [\![\langle q_i, s_1, s_2, d, q_j\rangle]\!] \longmapsto\!\!\!\!\longmapsto P$ for some P. Now by two applications of α-conversion it can be shown that $P \equiv (\nu l)(\nu r)(\ q_j s_x\langle l, r\rangle.\mathbf{0} \mid P_L \mid P_R \mid [\![\mathcal{F}]\!]\ )$ for some s_x and P_L and P_R. Now consider d.

- When $d = L$ it can be shown that $s_x = s_m$ and by two applications of induction on the indices of the restrictions (νx_i) of the left and right hand sides of the tape it can be shown that $P_L \equiv (\!|\ldots, s_k|\!)_L$ and that $P_R \equiv (\!|s_2, s_n, s_o, \ldots|\!)_R$.
- When $d = R$ is can be shown that $s_x = s_n$ and by two applications of induction on the indices of the restrictions (νx_i) of the left and right hand sides of the tape it can be shown that $P_L \equiv (\!|\ldots, s_k, s_m, s_2|\!)_L$ and that $P_R \equiv (\!|s_o, \ldots|\!)_R$.

The reverse direction is proved similarly by observing that the only possible reduction $[\![\langle\langle \mathcal{S},\mathcal{Q},\mathcal{F},\mathcal{T},q_i\rangle\rangle]\!] \longmapsto Q$ must be due to a tuple $\langle q_i,s_1,s_2,d,q_j\rangle$ that is in the transition function $\mathcal{F}$. Then it follows that there exists a reduction $Q \longmapsto Q'$ by definition of $[\![\langle q_i,s_1,s_2,d,q_j\rangle]\!]$. Finally showing that $Q' \equiv [\![\langle\langle \mathcal{S},\mathcal{Q},\mathcal{F},\mathcal{T}',q_j\rangle\rangle]\!]$ again requires tedious renaming of both sides of the encoded tape.

Theorem 6.4 *The encoding $[\![\cdot]\!]$ of a Turing Machine into (synchronous polyadic) π-calculus; preserves reduction, and divergence. That is, given a Turing Machine $\langle\langle \mathcal{S},\mathcal{Q},\mathcal{F},\mathcal{T},q_i\rangle\rangle$ then it holds that:*

1. *there is a transition $\langle\langle \mathcal{S},\mathcal{Q},\mathcal{F},\mathcal{T},q_i\rangle\rangle \longmapsto \langle\langle \mathcal{S},\mathcal{Q},\mathcal{F},\mathcal{T}',q_j\rangle\rangle$ if and only if there are reductions $[\![\langle\langle \mathcal{S},\mathcal{Q},\mathcal{F},\mathcal{T},q_i\rangle\rangle]\!] \longmapsto\!\!\longmapsto Q$ where $Q \equiv [\![\langle\langle \mathcal{S},\mathcal{Q},\mathcal{F},\mathcal{T}',q_j\rangle\rangle]\!]$, and*

2. *there is an infinite sequence of transitions $\langle\langle \mathcal{S},\mathcal{Q},\mathcal{F},\mathcal{T},q_i\rangle\rangle \longmapsto^{\omega}$ if and only if there is an infinite sequence of reductions $[\![\langle\langle \mathcal{S},\mathcal{Q},\mathcal{F},\mathcal{T},q_i\rangle\rangle]\!] \longmapsto^{\omega}$.*

7 Conclusions

The traditional approaches to encoding Turing Machines into process calculi tend to be indirect and lead to complex and unclear results. This is particularly true when the traditional path for process calculi is taken by encoding a Turing Machine into λ-calculus and then into a process calculus.

Recent calculi with intensional communication allow the representation of the current state and tape of a Turing Machine to be made clear and simple. Similarly, the capture of each transition of the Turing Machine by an input that transforms the state into a new output is a straightforward and elegant solution. The result is an encoding that is not only clearer and more direct, but also faithful and holds up to structural equivalence.

The encoding can also be adapted in various ways. The infinite terms of an infinite tape can be made finite if the tape of the Turing Machine has some finite sequence of symbols with infinite blanks on either side. The choice of asymmetric concurrent pattern calculus here is for clarity alone, the results also hold with only minor adaptations for both Concurrent Pattern Calculus and Psi Calculi.

The approach used here to encode Turing Machines into intensional process calculi can also be used to inform on similar approaches into non-intensional process calculi such as π-calculus. Although faithfulness is lost, the simplicity of intensional calculi in both: matching many names in a single interaction, and of binding complex structures to a single name, becomes clearer when observing the complexity required to use this approach in π-calculus. Thus despite π-calculi only losing faithfulness directly, the complexity of the encodings into π-calculi and having to work with many restrictions and renamings highlights the elegance of the encodings into intensional calculi.

Future Work

The rôle of intensionality in process calculi has not been explored in depth outside of particular calculi. A more general exploration of the expressiveness of intensionality remains to be published.

The approach of encoding Turing Machines by directly capturing the state and transitions has some similarities to the encoding of SF-logic [13] into CPC [9]. Adapting these approaches to other types of Turing Machines or rewriting systems is also of interest.

References

[1] Martín Abadi & Andrew D. Gordon (1997): *A Calculus for Cryptographic Protocols: The Spi Calculus*. In: *Proceedings of the 4th ACM Conference on Computer and Communications Security*, CCS '97, ACM, New York, NY, USA, pp. 36–47, doi:10.1145/266420.266432.

[2] Hendrik Pieter Barendregt (1984): *The Lambda calculus: Its syntax and semantics*. North-Holland, Amsterdam.

[3] Jesper Bengtson, Magnus Johansson, Joachim Parrow & Björn Victor (2011): *Psi-calculi: a framework for mobile processes with nominal data and logic*. *Logical Methods in Computer Science* 7(1), doi:10.2168/LMCS-7(1:11)2011.

[4] Gerard Berry & Gerard Boudol (1990): *The Chemical Abstract Machine*. In: *Proceedings of the 17th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, POPL '90, ACM, New York, NY, USA, pp. 81–94, doi:10.1145/96709.96717.

[5] G.S. Boolos, J.P. Burgess & R.C. Jeffrey (2007): *Computability and Logic*. Cambridge University Press, doi:10.1017/CBO9780511804076.

[6] Luca Cardelli & Andrew D. Gordon (1998): *Mobile Ambients*. In: *Foundations of Software Science and Computation Structures: First International Conference, FoSSaCS '98*, pp. 140–155, doi:10.1007/BFb0053547.

[7] H. B. Curry & R. Feys (1958): *Combinatory Logic*. I, North-Holland, Amsterdam.

[8] H. B. Curry, J. R. Hindley & J. P. Seldin (1972): *Combinatory Logic*. II, North-Holland, Amsterdam.

[9] Thomas Given-Wilson (2012): *Concurrent Pattern Unification*. PhD thesis, University of Technology, Sydney, Australia.

[10] Thomas Given-Wilson & Daniele Gorla (2013): *Pattern Matching and Bisimulation*. In Rocco De Nicola & Christine Julien, editors: *Coordination Models and Languages*, *Lecture Notes in Computer Science* 7890, Springer Berlin Heidelberg, pp. 60–74, doi:10.1007/978-3-642-38493-6_5.

[11] Thomas Given-Wilson, Daniele Gorla & Barry Jay (2010): *Concurrent Pattern Calculus*. In CristianS. Calude & Vladimiro Sassone, editors: *Theoretical Computer Science*, *IFIP Advances in Information and Communication Technology* 323, Springer Berlin Heidelberg, pp. 244–258, doi:10.1007/978-3-642-15240-5_18.

[12] Thomas Given-Wilson, Daniele Gorla & Barry Jay (2014): *A Concurrent Pattern Calculus*. *To appear in: Logical Methods in Computer Science*. Available at `http://hal.inria.fr/hal-00987578`.

[13] Barry Jay & Thomas Given-Wilson (2011): *A combinatory account of internal structure*. *Journal of Symbolic Logic* 76(3), pp. 807–826, doi:10.2178/jsl/1309952521.

[14] S.C. Kleene (1952): *Introduction to Metamathematics*. North-Holland (originally published by D. Van Nostrand).

[15] John McCarthy (1960): *Recursive Functions of Symbolic Expressions and Their Computation by Machine, Part I*. *Commun. ACM* 3(4), pp. 184–195, doi:10.1145/367177.367199.

[16] Robin Milner (1990): *Functions as processes*. In MichaelS. Paterson, editor: *Automata, Languages and Programming*, *Lecture Notes in Computer Science* 443, pp. 167–180, doi:10.1007/BFb0032030.

[17] Robin Milner (1999): *Communicating and mobile systems - the Pi-calculus*. Cambridge University Press.

[18] Robin Milner, Joachim Parrow & David Walker (1992): *A Calculus of Mobile Processes, I*. *Inf. Comput.* 100(1), pp. 1–40, doi:10.1016/0890-5401(92)90008-4.

[19] Robin Milner, Joachim Parrow & David Walker (1992): *A Calculus of Mobile Processes, II*. *Inf. Comput.* 100(1), pp. 41–77, doi:10.1016/0890-5401(92)90009-5.

[20] J. Parrow & B. Victor (1998): *The fusion calculus: expressiveness and symmetry in mobile processes*. In: *Logic in Computer Science, 1998. Proceedings. Thirteenth Annual IEEE Symposium on*, pp. 176–185, doi:10.1109/LICS.1998.705654.

[21] M. Schönfinkel (1924): *Über die Bausteine der mathematischen Logik*. *Mathematische Annalen* 92(3-4), pp. 305–316, doi:10.1007/BF01448013.

[22] A. Tarski (1956): *Logic, semantics, metamathematics*. In: *Intentions in Communication*, Oxford University Press, pp. 325–363.

[23] A. M. Turing (1936): *On Computable Numbers, with an application to the Entscheidungsproblem*. *Proceedings of the London Mathematical Society* 2(42), pp. 230–265, doi:10.1112/plms/s2-43.6.544.

[24] A. M. Turing (1937): *Computability and λ-definability*. *Journal of Symbolic Logic* 2, pp. 153–163, doi:10.2307/2268280.