Substructural Proofs as Automata

Henry DeYoung and Frank Pfenning

Carnegie Mellon University, Pittsburgh, PA 15213, USA
{hdeyoung,fp}@cs.cmu.edu

Abstract. We present subsingleton logic as a very small fragment of linear logic containing only $\oplus$, $\mathbf{1}$, least fixed points and allowing circular proofs. We show that cut-free proofs in this logic are in a Curry–Howard correspondence with subsequential finite state transducers. Constructions on finite state automata and transducers such as composition, complement, and inverse homomorphism can then be realized uniformly simply by cut and cut elimination. If we freely allow cuts in the proofs, they correspond to a well-typed class of machines we call *linear communicating automata*, which can also be seen as a generalization of Turing machines with multiple, concurrently operating read/write heads.

1 Introduction

In the early days of the study of computation as a discipline, we see fundamentally divergent models. On the one hand, we have Turing machines [16], and on the other we have Church's λ-calculus [4]. Turing machines are based on a finite set of states and an explicit storage medium (the tape) which can be read from, written to, and moved in small steps. The λ-calculus as a pure calculus of functions is founded on the notions of abstraction and composition, not easily available on Turing machines, and relies on the complex operation of substitution. The fact that they define the same set of computable functions, say, over natural numbers, is interesting, but are there deeper connections between Turing-like machine models of computation and Church-like linguistic models?

The discovery of the Curry–Howard isomorphism [5, 12] between intuitionistic natural deduction and the *typed* λ-calculus adds a new dimension. It provides a *logical foundation* for computation on λ-terms as a form of proof reduction. This has been tremendously important, as it has led to the development of type theory, the setting for much modern research in programming languages since design of a programming language and a logic for reasoning about its programs go hand in hand. To date, Turing-like machine models have not benefited from these developments since no clear and direct connections to logic along the lines of a Curry–Howard isomorphism were known.

In this paper, we explore several connections between certain kinds of automata and machines in the style of Turing and very weak fragments of linear logic [11] augmented with least fixed points along the lines of Baelde et al. [2] and Fortier and Santocanale [9]. Proofs are allowed to be circular with some conditions that ensure they can be seen as coinductively defined. We collectively

refer to these fragments as *subsingleton logic* because the rules naturally enforce that every sequent has at most one antecedent and succedent (Sect. 2).

Our first discovery is a Curry–Howard isomorphism between so-called *fixed-cut* proofs in $\oplus$,$\mathbf{1}$,μ-subsingleton logic and a slight generalization of deterministic finite-state transducers that also captures deterministic finite automata (Sects. 3 and 4). This isomorphism relates proofs to automata and proof reduction to state transitions of the automata. Constructions on automata such as composition, complement, and inverse homomorphism can then be realized "for free" on the logical side by a process of cut elimination (Sect. 5).

If we make two seemingly small changes – allowing arbitrary cuts instead of just fixed cuts and removing some restrictions on circular proofs – proof reduction already has the computational power of Turing machines. We can interpret proofs as a form of linear communicating automata (LCAs, Sect. 6), where *linear* means that the automata are lined up in a row and each automaton communicates only with its left and right neighbors. Alternatively, we can think of LCAs as a generalization of Turing machines with multiple read/write heads operating concurrently. LCAs can be subject to deadlock and race conditions, but those corresponding to (circular) proofs in $\oplus$,$\mathbf{1}$,μ-subsingleton logic do not exhibit these anomalies (Sect. 7). Thus, the logical connection defines well-behaved LCAs, analogous to the way natural deduction in intuitionistic implicational logic defines well-behaved λ-terms.

We also illustrate how traditional Turing machines are a simple special case of LCAs with only a single read/write head. Perhaps surprisingly, such LCAs can be typed and are therefore well-behaved by construction: Turing machines do not get stuck, while LCAs in general might (Sect. 7).

We view the results in this paper only as a beginning. Many natural questions remain. For example, can we capture deterministic pushdown automata or other classes of automata as natural fragments of the logic and its proofs? Can we exploit the logical origins beyond constructions by cut elimination to reason about properties of the automata or abstract machines?

2 A Subsingleton Fragment of Intuitionistic Linear Logic

In an intuitionistic linear sequent calculus, sequents consist of at most one conclusion in the context of zero or more hypotheses. To achieve a pleasant symmetry between contexts and conclusions, we can consider restricting contexts to have at most one hypothesis, so that each sequent has one of the forms $\cdot \vdash \gamma$ or $A \vdash \gamma$.

Is there a fragment of intuitionistic linear logic that obeys this rather harsh restriction and yet exists as a well-defined, interesting logic in its own right? Somewhat surprisingly, yes, there is; this section presents such a logic, which we dub $\oplus$,$\mathbf{1}$-subsingleton logic.

2.1 Propositions, Contexts, and Sequents

The propositions of $\oplus$,$\mathbf{1}$-subsingleton logic are generated by the grammar

$$A, B, C ::= A_1 \oplus A_2 \mid \mathbf{1}\,,$$

where $\oplus$ is additive disjunction and $\mathbf{1}$ is the unit of linear logic's multiplicative conjunction. Uninterpreted propositional atoms p could be included if desired, but we omit them because they are unnecessary for this paper's results. In Sect. 7, we will see that subsingleton logic can be expanded to include more, but not all, of the linear logical connectives.

Sequents are written $\Delta \vdash \gamma$. For now, we will have only single conclusions and so $\gamma ::= C$, but we will eventually consider empty conclusions in Sect. 7. To move toward a pleasant symmetry between contexts and conclusions, contexts Δ are empty or a single proposition, and so $\Delta ::= \cdot \mid A$. We say that a sequent obeys the *subsingleton context restriction* if its context adheres to this form.

2.2 Deriving the Inference Rules of $\oplus$,1-Subsingleton Logic

To illustrate how the subsingleton inference rules are derived from their counterparts in an intuitionistic linear sequent calculus, let us consider the cut rule. The subsingleton cut rule is derived from the intuitionistic linear cut rule as:

$$\frac{\Delta \vdash A \quad \Delta', A \vdash \gamma}{\Delta, \Delta' \vdash \gamma} \quad \rightsquigarrow \quad \frac{\Delta \vdash A \quad A \vdash \gamma}{\Delta \vdash \gamma}\ \textsc{cut}$$

In the original rule, the linear contexts Δ and Δ' may each contain zero or more hypotheses. When Δ' is nonempty, the sequent $\Delta', A \vdash \gamma$ fails to obey the subsingleton context restriction by virtue of using more than one hypothesis. But by dropping Δ' altogether, we derive a cut rule that obeys the restriction.

The other subsingleton inference rules are derived from linear counterparts in a similar way – just force each sequent to have a subsingleton context. Figure 1 summarizes the syntax and inference rules of a sequent calculus for $\oplus$,$\mathbf{1}$-subsingleton logic.

2.3 Admissibility of Cut and Identity

From the previous examples, we can see that it is not difficult to derive sequent calculus rules for $A_1 \oplus A_2$ and $\mathbf{1}$ that obey the subsingleton context restriction. But that these rules should constitute a well-defined logic in its own right is quite surprising!

Under the verificationist philosophies of Dummett [8] and Martin-Löf [13], $\oplus$,$\mathbf{1}$-subsingleton logic is indeed well-defined because it satisfies admissibility of CUT and ID, which characterize an internal soundness and completeness:

Theorem 1 (Admissibility of cut). *If there are proofs of $\Delta \vdash A$ and $A \vdash \gamma$, then there is also a cut-free proof of $\Delta \vdash \gamma$.*

Proof. By lexicographic induction, first on the structure of the cut formula A and then on the structures of the given derivations.

Theorem 2 (Admissibility of identity). *For all propositions A, the sequent $A \vdash A$ is derivable without using* ID.

$$\begin{array}{rl} \text{Propositions} & A, B, C ::= A_1 \oplus A_2 \mid \mathbf{1} \\ \text{Contexts} & \Delta ::= \cdot \mid A \\ \text{Conclusions} & \gamma ::= C \end{array}$$

$$\frac{}{A \vdash A}\ \textsc{id} \qquad \frac{\Delta \vdash A \quad A \vdash \gamma}{\Delta \vdash \gamma}\ \textsc{cut}$$

$$\frac{\Delta \vdash A_1}{\Delta \vdash A_1 \oplus A_2}\ \oplus\text{R}_1 \qquad \frac{\Delta \vdash A_2}{\Delta \vdash A_1 \oplus A_2}\ \oplus\text{R}_2$$

$$\frac{A_1 \vdash \gamma \quad A_2 \vdash \gamma}{A_1 \oplus A_2 \vdash \gamma}\ \oplus\text{L}$$

$$\frac{}{\cdot \vdash \mathbf{1}}\ \mathbf{1}\text{R} \qquad \frac{\cdot \vdash \gamma}{\mathbf{1} \vdash \gamma}\ \mathbf{1}\text{L}$$

Fig. 1. A sequent calculus for $\oplus$,$\mathbf{1}$-subsingleton logic

Proof. By structural induction on A.

Theorem 2 justifies hereafter restricting our attention to a calculus without the ID rule. The resulting proofs are said to be identity-free, or η-long, and are complete for provability. Despite Theorem 1, we do not restrict our attention to cut-free proofs because the CUT rule will prove to be important for composition of machines.

2.4 Extending the Logic with Least Fixed Points

Thus far, we have presented a sequent calculus for $\oplus$,$\mathbf{1}$-subsingleton logic with finite propositions $A_1 \oplus A_2$ and $\mathbf{1}$. Now we extend it with least fixed points $\mu\alpha.A$, keeping an eye toward their eventual Curry–Howard interpretation as the types of inductively defined data structures. We dub the extended logic $\oplus$,$\mathbf{1}$,μ-subsingleton logic.

Our treatment of least fixed points mostly follows that of Fortier and Santocanale [9] by using circular proofs. Here we review the intuition behind circular proofs; please refer to Fortier and Santocanale's publication for a full, formal description.

Fixed Point Propositions and Sequents. Syntactically, the propositions are extended to include least fixed points $\mu\alpha.A$ and propositional variables α:

$$A, B, C ::= \cdots \mid \mu\alpha.A \mid \alpha$$

Because the logic's propositional connectives – just $\oplus$ and $\mathbf{1}$ for now – are all covariant, least fixed points necessarily satisfy the usual strict positivity condition that guarantees well-definedness. We also require that least fixed points are

contractive [10], ruling out, for example, $\mu\alpha.\alpha$. Finally, we further require that a sequent's hypothesis and conclusion be closed, with no free occurrences of any propositional variables α.

In a slight departure from Fortier and Santocanale, we treat least fixed points *equirecursively*, so that $\mu\alpha.A$ is identified with its unfoldings, $[(\mu\alpha.A)/\alpha]A$ and so on. When combined with contractivity, this means that $\mu\alpha.A$ may be thought of as a kind of infinite proposition. For example, $\mu\alpha.\,\mathbf{1} \oplus \alpha$ is something like $\mathbf{1} \oplus (\mathbf{1} \oplus \cdots)$.

Circular Proofs. Previously, with only finite propositions and inference rules that obeyed a subformula property, proofs in $\oplus$,**1**-subsingleton logic were the familiar well-founded trees of inferences. Least fixed points could be added to this finitary sequent calculus along the lines of Baelde's μMALL [1], but it will be more convenient and intuitive for us to follow Fortier and Santocanale and use an infinitary sequent calculus of circular proofs.

To illustrate the use of circular proofs, consider the following proof, which has as its computational content the function that doubles a natural number. Natural numbers are represented as proofs of the familiar least fixed point $\mathtt{Nat} = \mu\alpha.\,\mathbf{1} \oplus \alpha$; the unfolding of $\mathtt{Nat}$ is thus $\mathbf{1} \oplus \mathtt{Nat}$.

$$\dfrac{\dfrac{\dfrac{\dfrac{}{\cdot \vdash \mathbf{1}}\,\mathbf{1}\mathrm{R}}{\cdot \vdash \mathbf{1} \oplus \mathtt{Nat}}\,\oplus\mathrm{R}_1 \qquad \dfrac{\dfrac{\mathtt{Nat} \vdash \mathtt{Nat}}{\mathtt{Nat} \vdash \mathbf{1} \oplus \mathtt{Nat}}\,\oplus\mathrm{R}_2}{\dfrac{\mathtt{Nat} \vdash \mathtt{Nat}}{\mathtt{Nat} \vdash \mathbf{1} \oplus \mathtt{Nat}}\,\oplus\mathrm{R}_2}}{\dfrac{\dfrac{\cdot \vdash \mathtt{Nat}}{\mathbf{1} \vdash \mathtt{Nat}}\,\mathbf{1}\mathrm{L} \qquad \mathtt{Nat} \vdash \mathtt{Nat}}{\mathbf{1} \oplus \mathtt{Nat} \vdash \mathtt{Nat}}\,\oplus\mathrm{L}}}{\mathtt{Nat} \vdash \mathtt{Nat}} \tag{1}$$

This proof begins by case-analyzing a $\mathtt{Nat}$ ($\oplus$L rule). If the number is 0, then the proof's left branch continues by reconstructing 0. Otherwise, if the number is the successor of some natural number N, then the proof's right branch continues by first emitting two successors ($\oplus\mathrm{R}_2$ rules) and then making a recursive call to double N, as indicated by the back-edge drawn with an arrow.

In this proof, there are several instances of unfolding $\mathtt{Nat}$ to $\mathbf{1} \oplus \mathtt{Nat}$. In general, the principles for unfolding on the right and left of a sequent are

$$\dfrac{\Delta \vdash [(\mu\alpha.A)/\alpha]}{\Delta \vdash \mu\alpha.A} \qquad \text{and} \qquad \dfrac{[(\mu\alpha.A)/\alpha] \vdash \gamma}{\mu\alpha.A \vdash \gamma}$$

Fortier and Santocanale adopt these principles as primitive right and left rules for μ. But because our least fixed points are equirecursive and a fixed point is *equal* to its unfolding, unfolding is not a first-class rule of inference, but rather a principle that is used silently within a proof. It would thus be more accurate, but also more opaque, to write the above proof without those dotted principles.

Is μ Correctly Defined? With proofs being circular and hence coinductively defined, one might question whether $\mu\alpha.A$ really represents a *least* fixed point

and not a greatest fixed point. After all, we have no inference rules for μ, only implicit unfolding principles – and those principles could apply to any fixed points, not just least ones.

Stated differently, how do we proscribe the following, which purports to represent the first transfinite ordinal, ω, as a finite natural number?

$$\cfrac{\cdot \vdash \mathtt{Nat}}{\cdot \vdash \mathbf{1} \oplus \mathtt{Nat}}\ \oplus\mathrm{R}_2$$
$$\cdot \vdash \mathtt{Nat}$$

To ensure that μ is correctly defined, one last requirement is imposed upon valid proofs: that every cycle in a valid proof is a left μ-trace. A left μ-trace *(i)* contains at least one application of a left rule to the unfolding of a least fixed point hypothesis, and *(ii)* if the trace contains an application of the CUT rule, then the trace continues along the left premise of the CUT. The above $\mathtt{Nat} \vdash \mathtt{Nat}$ example is indeed a valid proof because its cycle applies the $\oplus$L rule to $\mathbf{1} \oplus \mathtt{Nat}$, the unfolding of a $\mathtt{Nat}$ hypothesis. But the attempt at representing ω is correctly proscribed because its cycle contains no least fixed point hypothesis whatsoever, to say nothing of a left rule.

Cut Elimination for Circular Proofs. Fortier and Santocanale [9] present a cut elimination procedure for circular proofs. Because of their infinitary nature, circular proofs give rise to a different procedure than do the familiar finitary proofs.

Call a circular proof a *fixed-cut* proof if no cycle contains the CUT rule. Notice the subtle difference from cut-free circular proofs – a fixed-cut proof may contain the CUT rule, so long as the cut occurs outside of all cycles. Cut elimination on *fixed-cut* circular proofs results in a cut-free circular proof.

Things are not quite so pleasant for cut elimination on arbitrary circular proofs. In general, cut elimination results in an infinite, cut-free proof that is not necessarily circular.

3 Subsequential Finite-State Transducers

Subsequential finite-state transducers (SFTs) were first proposed by Schützenberger [15] as a way to capture a class of functions from finite strings to finite strings that is related to finite automata and regular languages. An SFT T is fed some string w as input and deterministically produces a string v as output.

Here we review one formulation of SFTs. This formulation classifies each SFT state as reading, writing, or halting so that SFT computation occurs in small, single-letter steps. Also, this formulation uses strings over alphabets with (potentially several) endmarker symbols so that a string's end is apparent from its structure and so that SFTs subsume deterministic finite automata (Sect. 3.3). Lastly, this formulation uses string reversal in a few places so that SFT configurations receive their input from the left and produce output to the right.

In later sections, we will see that these SFTs are isomorphic to a class of cut-free proofs in subsingleton logic.

3.1 Definitions

Preliminaries. As usual, the set of all finite strings over an alphabet Σ is written as Σ^*, with ϵ denoting the empty string. In addition, the reversal of a string $w \in \Sigma^*$ is written $w^{\mathcal{R}}$.

An *endmarked alphabet* is a pair $\hat{\Sigma} = (\Sigma_{\mathsf{i}}, \Sigma_{\mathsf{e}})$, consisting of disjoint finite alphabets Σ_{i} and Σ_{e} of internal symbols and endmarkers, respectively, with Σ_{e} nonempty. Under the endmarked alphabet $\hat{\Sigma}$, the set of finite strings terminated with an endmarker is $\Sigma_{\mathsf{i}}^* \Sigma_{\mathsf{e}}$, which we abbreviate as $\hat{\Sigma}^+$. It will be convenient to also define $\hat{\Sigma}^* = \hat{\Sigma}^+ \cup \{\epsilon\}$ and $\Sigma = \Sigma_{\mathsf{i}} \cup \Sigma_{\mathsf{e}}$.

Subsequential Transducers. A *subsequential finite-state string transducer (SFT)* is a 6-tuple $T = (Q, \hat{\Sigma}, \hat{\Gamma}, \delta, \sigma, q_0)$ where Q is a finite set of states that is partitioned into (possibly empty) sets of read and write states, Q^{r} and Q^{w}, and halt states, Q^{h}; $\hat{\Sigma} = (\Sigma_{\mathsf{i}}, \Sigma_{\mathsf{e}})$ with $\Sigma_{\mathsf{e}} \neq \emptyset$ is a finite endmarked alphabet for input; $\hat{\Gamma} = (\Gamma_{\mathsf{i}}, \Gamma_{\mathsf{e}})$ with $\Gamma_{\mathsf{e}} \neq \emptyset$ is a finite endmarked alphabet for output; $\delta \colon \Sigma \times Q^{\mathsf{r}} \to Q$ is a total transition function on read states; $\sigma \colon Q^{\mathsf{w}} \to Q \times \Gamma$ is a total output function on write states; and $q_0 \in Q$ is the initial state.

Configurations $\mathcal{C}$ of the SFT T have one of two forms – either *(i)* $w\,q\,v$, where $w^{\mathcal{R}} \in \hat{\Sigma}^*$ and $q \in Q$ and $v^{\mathcal{R}} \in (\Gamma_{\mathsf{i}}^* \cup \hat{\Gamma}^*)$; or *(ii)* v, where $v^{\mathcal{R}} \in \hat{\Gamma}^+$. Let $\longrightarrow$ be the least binary relation on configurations that satisfies the following conditions.

$$
\begin{array}{lrll}
\textsc{read} & wa\,q\,v & \longrightarrow w\,q_a\,v & \text{if } q \in Q^{\mathsf{r}} \text{ and } \delta(a,q) = q_a \\
\textsc{write} & w\,q\,v & \longrightarrow w\,q_b\,bv & \text{if } q \in Q^{\mathsf{w}} \text{ and } \sigma(q) = (q_b, b) \text{ and } v \in \Gamma_{\mathsf{i}}^* \\
\textsc{halt} & q\,v & \longrightarrow v & \text{if } q \in Q^{\mathsf{h}} \text{ and } v^{\mathcal{R}} \in \hat{\Gamma}^+
\end{array}
$$

The SFT T is said to *transduce input* $w \in \hat{\Sigma}^+$ *to output* $v \in \hat{\Gamma}^+$ if there exists a sequence of configurations $\mathcal{C}_0, \ldots, \mathcal{C}_n$ such that *(i)* $\mathcal{C}_0 = w^{\mathcal{R}} q_0$; *(ii)* $\mathcal{C}_i \longrightarrow \mathcal{C}_{i+1}$ for all $0 \leq i < n$; and *(iii)* $\mathcal{C}_n = v^{\mathcal{R}}$.

3.2 Example of a Subsequential Transducer

Figure 2 shows the transition graph for an SFT over $\hat{\Sigma} = (\{a, b\}, \{\$\})$. The edges in this graph are labeled c or $\overline{c}$ to indicate an input or output of symbol c, respectively. This SFT compresses each run of bs into a single b. For instance, the input string $abbaabbb\$$ transduces to the output string $abaab\$$ because $\$bbbaabba\,q_0 \longrightarrow^+ \$baaba$. We could even compose this SFT with itself, but this SFT is an idempotent for composition.

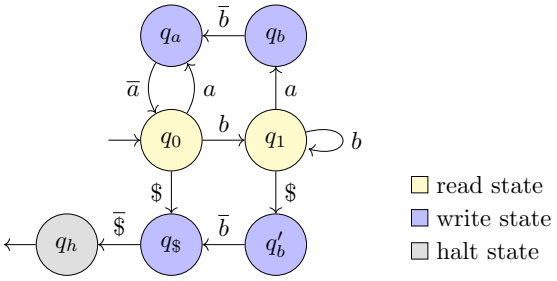

Fig. 2. A subsequential finite-state transducer over the endmarked alphabet $\hat{\Sigma} = (\{a, b\}, \{\$\})$ that compresses each run of bs into a single b

3.3 Discussion

Acceptance and Totality. Notice that, unlike some definitions of SFTs, this definition does not include notions of acceptance or rejection of input strings. This is because we are interested in SFTs that induce a total transduction function, since such transducers turn out to compose more naturally in our proof-theoretic setting.

Normal Form SFTs. The above formulation of SFTs allows the possibility that a read state is reachable even after an endmarker signaling the end of the input has been read. An SFT would necessarily get stuck upon entering such a state because there is no more input to read.

The above formulation also allows the dual possibility that a write state is reachable even after having written an endmarker signaling the end of the output. Again, an SFT would necessarily get stuck upon entering such a state because the side condition of the WRITE rule, $v \in \Gamma_{\mathsf{i}}^*$, would fail to be met.

Lastly, the above formulation allows that a halt state is reachable before an endmarker signaling the end of the input has been read. According to the HALT rule, an SFT would necessarily get stuck upon entering such a state.

Fortunately, we may define *normal-form SFTs* as SFTs for which these cases are impossible. An SFT is in normal form if it obeys three properties:

- For all endmarkers $e \in \Sigma_{\mathsf{e}}$ and read states $q \in Q^{\mathsf{r}}$, no read state is reachable from $\delta(e, q)$.
- For all endmarkers $e \in \Gamma_{\mathsf{e}}$, write states $q \in Q^{\mathsf{w}}$, and states $q_e \in Q$, no write state is reachable from q_e if $\sigma(q) = (q_e, e)$.
- For all halt states $q \in Q^{\mathsf{w}}$, all paths from the initial state q_0 to q pass through $\delta(e, q')$ for some endmarker $e \in \Sigma_{\mathsf{e}}$ and read state $q' \in Q^{\mathsf{r}}$.

Normal-form SFTs and SFTs differ only on stuck computations. Because we are only interested in total transductions, hereafter we assume that all SFTs are normal-form.

Deterministic Finite Automata. By allowing alphabets with more than one endmarker, the above definition of SFTs subsumes deterministic finite automata (DFAs). A DFA is an SFT with an endmarked output alphabet $\hat{\Gamma} = (\emptyset, \{a, r\})$, so that the valid output strings are only a or r; the DFA transduces its input to the output string a or r to indicate acceptance or rejection of the input, respectively.

3.4 Composing Subsequential Finite-State String Transducers

Having considered individual subsequential finite-state transducers (SFTs), we may want to compose finitely many SFTs into a linear network that implements a transduction in a modular way. Fortunately, in the above model, SFTs and their configurations compose very naturally into chains.

An SFT chain $(T_i)_{i=1}^n$ is a finite family of SFTs $T_i = (Q_i, \hat{\Sigma}_i, \hat{\Gamma}_i, \delta_i, \sigma_i, q_i)$ such that $\hat{\Gamma}_i = \hat{\Sigma}_{i+1}$ for each $i < n$. Here we give a description of the special case $n = 2$; the general case is notationally cumbersome without providing additional insight.

Let $T_1 = (Q_1, \hat{\Sigma}, \hat{\Gamma}, \delta_1, \sigma_1, i_1)$ and $T_2 = (Q_2, \hat{\Gamma}, \hat{\Omega}, \delta_2, \sigma_2, i_2)$ be two SFTs; let $\hat{\Sigma}_1 = \hat{\Sigma}$ and $\hat{\Gamma}_1 = \hat{\Sigma}_2 = \hat{\Gamma}$ and $\hat{\Gamma}_2 = \hat{\Omega}$. A configuration of the chain $(T_i)_{i=1}^2$ is a string whose reversal is drawn from either $(\Omega_{\mathsf{i}}^* \cup \hat{\Omega}^*)\, Q_2\, (\Gamma_{\mathsf{i}}^* \cup \hat{\Gamma}^*)\, Q_1\, \hat{\Sigma}^*$ or $(\Omega_{\mathsf{i}}^* \cup \hat{\Omega}^*)\, Q_2\, \hat{\Gamma}^*$ or $\hat{\Omega}^+$. Let $\longrightarrow$ be the least binary relation on configurations that satisfies the following conditions.

$$
\begin{array}{lrll}
\textsc{read} & w a\, q_i\, v & \longrightarrow w\, q_i'\, v & \text{if } \delta_i(a, q_i) = q_i' \\
\textsc{write} & w\, q_i\, v & \longrightarrow w\, q_i'\, b v & \text{if } \sigma_i(q_i) = (q_i', b) \\
\textsc{halt} & q_i\, v & \longrightarrow v & \text{if } q_i \in Q_i^{\mathsf{h}} \text{ and } v \text{ is a config.}
\end{array}
$$

Thus, composition of SFTs is accomplished by concatenating the states of the individual SFTs. The composition of T_1 and T_2 transduces $w \in \hat{\Sigma}^+$ to $v \in \hat{\Omega}^+$ if $w^{\mathcal{R}}\, i_1\, i_2 \longrightarrow^* v^{\mathcal{R}}$.

Notice that an asynchronous, concurrent semantics of transducer composition comes for free with this model. For example, in the transducer chain $w\, q_1\, q_2\, \cdots\, q_n$, the state q_1 can react to the next symbol of input while q_2 is still absorbing q_1's first round of output.

4 Curry–Howard Isomorphism for Subsingleton Proofs

In this section, we turn our attention from a machine model of subsequential finite state transducers (SFTs) to a computational interpretation of the $\oplus$,$\mathbf{1}$,μ-subsingleton sequent calculus. We then bridge the two by establishing a Curry–Howard isomorphism between SFTs and a class of cut-free subsingleton proofs – propositions are languages, proofs are SFTs, and cut reductions are SFT computation steps. In this way, the cut-free proofs of subsingleton logic serve as a linguistic model that captures exactly the subsequential functions.

$$
\begin{array}{rl}
\text{Types} & A, B, C ::= \oplus_{\ell \in L}\{\ell{:}A_\ell\} \mid \mathbf{1} \mid \mu\alpha.A \mid \alpha \\
\text{Contexts} & \Delta ::= \cdot \mid A \\
\text{Conclusions} & \gamma ::= C \\
\text{Proof terms} & P, Q ::= X \mid P \triangleright Q \\
 & \qquad \mid \mathsf{writeR}\ k; P \mid \mathsf{readL}_{\ell \in L}(\ell \Rightarrow Q_\ell) \\
 & \qquad \mid \mathsf{closeR} \mid \mathsf{waitL}; Q \\
\text{Signatures} & \Theta ::= \cdot \mid \Theta, (\Delta \vdash X = P : \gamma)
\end{array}
$$

$$
\frac{(\Delta \vdash X = P : \gamma) \in \Theta}{\Delta \vdash_\Theta X : \gamma}\ \textsc{var} \qquad \frac{\Delta \vdash_\Theta P : A \quad A \vdash_\Theta Q : \gamma}{\Delta \vdash_\Theta P \triangleright Q : \gamma}\ \textsc{cut} \qquad \text{(no ID rule)}
$$

$$
\frac{\Delta \vdash_\Theta P : A_k \quad (k \in L)}{\Delta \vdash_\Theta \mathsf{writeR}\ k; P : \oplus_{\ell \in L}\{\ell{:}A_\ell\}}\ \oplus\textsc{r} \qquad \frac{\forall \ell \in L\colon\ A_\ell \vdash_\Theta Q_\ell : \gamma}{\oplus_{\ell \in L}\{\ell{:}A_\ell\} \vdash_\Theta \mathsf{readL}_{\ell \in L}(\ell \Rightarrow Q_\ell) : \gamma}\ \oplus\textsc{l}
$$

$$
\frac{}{\cdot \vdash_\Theta \mathsf{closeR} : \mathbf{1}}\ \mathbf{1}\textsc{r} \qquad \frac{\cdot \vdash_\Theta Q : \gamma}{\mathbf{1} \vdash_\Theta \mathsf{waitL}; Q : \gamma}\ \mathbf{1}\textsc{l}
$$

$$
\frac{\Delta \vdash_\Theta P : [(\mu\alpha.A)/\alpha]A}{\Delta \vdash_\Theta P : \mu\alpha.A} \qquad \frac{[(\mu\alpha.A)/\alpha]A \vdash_\Theta Q : C}{\mu\alpha.A \vdash_\Theta Q : C}
$$

$$
\frac{}{\vdash_{\Theta'} (\cdot)\ \mathsf{ok}}\ \textsc{ok-e} \qquad \frac{\vdash_{\Theta'} \Theta\ \mathsf{ok} \quad \Delta \vdash_{\Theta'} P : \gamma}{\vdash_{\Theta'} \Theta, (\Delta \vdash X = P : \gamma)\ \mathsf{ok}}\ \textsc{ok-var}
$$

$$
\begin{array}{c}
(\mathsf{writeR}\ k; P) \triangleright \mathsf{readL}_{\ell \in L}(\ell \Rightarrow Q_\ell) \longrightarrow P \triangleright Q_k \\
\mathsf{closeR} \triangleright (\mathsf{waitL}; Q) \longrightarrow Q
\end{array}
$$

Fig. 3. A proof term assignment and the principal cut reductions for the $\oplus$,$\mathbf{1}$,μ-subsingleton sequent calculus

4.1 A Computational Interpretation of $\oplus$,1,μ-Subsingleton Logic

Figure 3 summarizes our computational interpretation of the $\oplus$,$\mathbf{1}$,μ-subsingleton sequent calculus.

Now that we are emphasizing the logic's computational aspects, it will be convenient to generalize binary additive disjunctions to n-ary, labeled additive disjunctions, $\oplus_{\ell \in L}\{\ell{:}A_\ell\}$. We require that the set L of labels is nonempty, so that n-ary, labeled additive disjunction does not go beyond what may be expressed (less concisely) with the binary form.[1] Thus, propositions are now generated by the grammar

$$A, B, C ::= \oplus_{\ell \in L}\{\ell{:}A_\ell\} \mid \mathbf{1} \mid \mu\alpha.A \mid \alpha\,.$$

[1] Notice that the proposition $\oplus\{k{:}A\}$ is distinct from A.

Contexts Δ still consist of exactly zero or one proposition and conclusions γ are still single propositions. Each sequent $\Delta \vdash \gamma$ is now annotated with a proof term P and a signature Θ, so that $\Delta \vdash_\Theta P : \gamma$ is read as "Under the definitions of signature Θ, the proof term P consumes input of type Δ to produce output of type γ." Already, the proof term P sounds vaguely like an SFT.

The logic's inference rules now become typing rules for proof terms. The $\oplus$R rule types a write operation, $\mathsf{writeR}\ k; P$, that emits label k and then continues; dually, the $\oplus$L rule types a read operation, $\mathsf{readL}_{\ell \in L}(\ell \Rightarrow Q_\ell)$, that branches on the label that was read. The $\mathbf{1}$R rule types an operation, closeR, that signals the end of the output; the $\mathbf{1}$L rule types an operation, $\mathsf{waitL}; Q$, that waits for the input to end and then continues with Q. The CUT rule types a composition, $P \triangleright Q$, of proof terms P and Q. Lastly, unfolding principles are used silently within a proof and do not affect the proof term.

The circularities inherent to circular proofs are expressed with a finite signature Θ of mutually corecursive definitions. Each definition in Θ has the form $\Delta \vdash X = P : \gamma$, defining the variable X as proof term P with a type declaration of $\Delta \vdash_\Theta X : \gamma$. We rule out definitions of the forms $X = X$ and $X = Y$. To verify that the definitions in Θ are well-typed, we check that $\vdash_\Theta \Theta\ \mathsf{ok}$ according to the rules given in Fig. 3. Note that the same signature Θ' (initially Θ) is used to type all variables, which thereby allows arbitrary mutual recursion.

As an example, here are two well-typed definitions:

$$
\begin{aligned}
X_0 = \mathsf{caseL}(&a \Rightarrow \mathsf{writeR}\ a; X_0 \\
&\mid b \Rightarrow X_1 \\
&\mid \$ \Rightarrow \mathsf{waitL}; \\
&\qquad \mathsf{writeR}\ \$; \mathsf{closeR})
\end{aligned}
\qquad\qquad
\begin{aligned}
X_1 = \mathsf{caseL}(&a \Rightarrow \mathsf{writeR}\ b; \mathsf{writeR}\ a; X_0 \\
&\mid b \Rightarrow X_1 \\
&\mid \$ \Rightarrow \mathsf{waitL}; \mathsf{writeR}\ b; \\
&\qquad \mathsf{writeR}\ \$; \mathsf{closeR})
\end{aligned}
$$

4.2 Propositions as Languages

Here we show that propositions are languages over finite endmarked alphabets. However, before considering all freely generated propositions, let us look at one in particular: the least fixed point $\mathtt{Str}_{\hat{\Sigma}} = \mu\alpha. \oplus_{\ell \in \Sigma} \{\ell{:}A_\ell\}$ where $A_a = \alpha$ for all $a \in \Sigma_{\mathsf{i}}$ and $A_e = \mathbf{1}$ for all $e \in \Sigma_{\mathsf{e}}$. By unfolding,

$$\mathtt{Str}_{\hat{\Sigma}} = \oplus_{\ell \in \Sigma} \{\ell{:}A'_\ell\}\text{, where } A'_\ell = \begin{cases} \mathtt{Str}_{\hat{\Sigma}} & \text{if } \ell \in \Sigma_{\mathsf{i}} \\ \mathbf{1} & \text{if } \ell \in \Sigma_{\mathsf{e}} \end{cases}$$

The proposition $\mathtt{Str}_{\hat{\Sigma}}$ is a type that describes the language $\hat{\Sigma}^+$ of all finite strings over the endmarked alphabet $\hat{\Sigma}$.

Theorem 3. *Strings from the language $\hat{\Sigma}^+$ are in bijective correspondence with the cut-free proofs of $\cdot \vdash \mathtt{Str}_{\hat{\Sigma}}$.*

A cut-free proof term P of type $\cdot \vdash \mathtt{Str}_{\hat{\Sigma}}$ emits a finite list of symbols from $\hat{\Sigma}$. By inversion on its typing derivation, P is either: $\mathsf{writeR}\ e; \mathsf{closeR}$, which terminates the list by emitting some endmarker $e \in \Sigma_{\mathsf{e}}$; or $\mathsf{writeR}\ a; P'$, which continues the list by emitting some symbol $a \in \Sigma_{\mathsf{i}}$ and then behaving as proof term P' of

type $\cdot \vdash \mathtt{Str}_{\hat{\Sigma}}$. The above intuition can be made precise by defining a bijection $[\![-]\!] \colon \hat{\Sigma}^+ \to (\cdot \vdash \mathtt{Str}_{\hat{\Sigma}})$ along these lines. As an example, the string $ab\$ \in \hat{\Sigma}^+$ with $\hat{\Sigma} = (\{a, b\}, \{\$\})$ corresponds to $[\![ab\$]\!] = \mathsf{writeR}\ a; \mathsf{writeR}\ b; \mathsf{writeR}\ \$; \mathsf{closeR}$.

The freely generated propositions correspond to subsets of $\hat{\Sigma}^+$. This can be seen most clearly if we introduce subtyping [10], but we do not do so because we are interested only in $\mathtt{Str}_{\hat{\Sigma}}$ hereafter.

4.3 Encoding SFTs as Cut-Free Proofs

Having now defined a type $\mathtt{Str}_{\hat{\Sigma}}$ and shown that $\hat{\Sigma}^+$ is isomorphic to cut-free proofs of $\cdot \vdash \mathtt{Str}_{\hat{\Sigma}}$, we can now turn to encoding SFTs as proofs. We encode each of the SFT's states as a cut-free proof of $\mathtt{Str}_{\hat{\Sigma}} \vdash \mathtt{Str}_{\hat{\Gamma}}$; this proof captures a (subsequential) function on finite strings.

Let $T = (Q, \hat{\Sigma}, \hat{\Gamma}, \delta, \sigma, q_0)$ be an arbitrary SFT in normal form. Define a mutually corecursive family of definitions $[\![q]\!]_T$, one for each state $q \in Q$. There are three cases according to whether q is a read, a write, or a halt state.

- If q is a read state, then $[\![q]\!] = \mathsf{readL}_{a \in \Sigma}(a \Rightarrow P_a)$, where for each a

$$P_a = \begin{cases} [\![q_a]\!] & \text{if } a \in \Sigma_{\mathsf{i}} \text{ and } \delta(a, q) = q_a \\ \mathsf{waitL}; [\![q_a]\!] & \text{if } a \in \Sigma_{\mathsf{e}} \text{ and } \delta(a, q) = q_a \end{cases}$$

  When q is reachable from some state q' that writes an endmarker, we declare $[\![q]\!]$ to have type $\mathtt{Str}_{\hat{\Sigma}} \vdash [\![q]\!] : \mathbf{1}$. Otherwise, we declare $[\![q]\!]$ to have type $\mathtt{Str}_{\hat{\Sigma}} \vdash [\![q]\!] : \mathtt{Str}_{\hat{\Gamma}}$.
- If q is a write state such that $\sigma(q) = (q_b, b)$, then $[\![q]\!] = \mathsf{writeR}\ b; [\![q_b]\!]$. When q is reachable from $\delta(e, q')$ for some $e \in \Sigma_{\mathsf{e}}$ and $q' \in Q^{\mathsf{r}}$, we declare $[\![q]\!]$ to have type $\cdot \vdash \mathtt{Str}_{\hat{\Gamma}}$. Otherwise, we declare $[\![q]\!]$ to have type $\mathtt{Str}_{\hat{\Sigma}} \vdash \mathtt{Str}_{\hat{\Gamma}}$.
- If q is a halt state, then $[\![q]\!] = \mathsf{closeR}$. This definition has type $\cdot \vdash [\![q]\!] : \mathbf{1}$.

When the SFT is in normal form, these definitions are well-typed. A type declaration with an empty context indicates that an endmarker has already been read. Because the reachability condition on read states in normal-form SFTs proscribes read states from occurring once an endmarker has been read, the type declarations $\mathtt{Str}_{\hat{\Sigma}} \vdash \mathtt{Str}_{\hat{\Gamma}}$ or $\mathtt{Str}_{\hat{\Sigma}} \vdash \mathbf{1}$ for read states is valid. Because normal-form SFTs also ensure that halt states only occur once an endmarker has been read, the type declaration $\cdot \vdash \mathbf{1}$ for halt states is valid.

As an example, the SFT from Fig. 2 can be encoded as follows.

$$\mathtt{Str}_{\hat{\Sigma}} = \oplus\{a{:}\mathtt{Str}_{\hat{\Sigma}}, b{:}\mathtt{Str}_{\hat{\Sigma}}, \${:}\mathbf{1}\}$$

$$\begin{array}{l} \mathtt{Str}_{\hat{\Sigma}} \vdash [\![q_0]\!] : \mathtt{Str}_{\hat{\Sigma}} \\ [\![q_0]\!] = \mathsf{readL}(a \Rightarrow [\![q_a]\!] \mid b \Rightarrow [\![q_1]\!] \\ \qquad\qquad \mid \$ \Rightarrow \mathsf{waitL}; [\![q_\$]\!]) \end{array} \qquad \begin{array}{l} \mathtt{Str}_{\hat{\Sigma}} \vdash [\![q_1]\!] : \mathtt{Str}_{\hat{\Sigma}} \\ [\![q_1]\!] = \mathsf{readL}(a \Rightarrow [\![q_b]\!] \mid b \Rightarrow [\![q_1]\!] \\ \qquad\qquad \mid \$ \Rightarrow \mathsf{waitL}; [\![q'_b]\!]) \end{array}$$

$$\begin{array}{l} \mathtt{Str}_{\hat{\Sigma}} \vdash [\![q_a]\!], [\![q_b]\!] : \mathtt{Str}_{\hat{\Sigma}} \\ [\![q_a]\!] = \mathsf{writeR}\ a; [\![q_0]\!] \\ [\![q_b]\!] = \mathsf{writeR}\ b; [\![q_a]\!] \end{array} \qquad \begin{array}{l} \cdot \vdash [\![q'_b]\!], [\![q_\$]\!] : \mathtt{Str}_{\hat{\Sigma}} \\ [\![q'_b]\!] = \mathsf{writeR}\ b; [\![q_\$]\!] \\ [\![q_\$]\!] = \mathsf{writeR}\ \$; [\![q_h]\!] \end{array} \qquad \begin{array}{l} \cdot \vdash [\![q_h]\!] : \mathbf{1} \\ [\![q_h]\!] = \mathsf{closeR} \end{array}$$

If one doesn't care about a bijection between definitions and states, some of these definitions can be folded into $[\![q_0]\!]$ and $[\![q_1]\!]$.

$$\begin{array}{l}\mathtt{Str}_{\hat{\Sigma}} \vdash [\![q_0]\!] : \mathtt{Str}_{\hat{\Sigma}} \\ [\![q_0]\!] = \mathsf{caseL}(a \Rightarrow \mathsf{writeR}\ a; [\![q_0]\!] \\ \qquad\qquad \mid b \Rightarrow [\![q_1]\!] \\ \qquad\qquad \mid \$ \Rightarrow \mathsf{waitL}; \\ \qquad\qquad\qquad \mathsf{writeR}\ \$; \mathsf{closeR})\end{array} \qquad \begin{array}{l}\mathtt{Str}_{\hat{\Sigma}} \vdash [\![q_1]\!] : \mathtt{Str}_{\hat{\Sigma}} \\ [\![q_1]\!] = \mathsf{caseL}(a \Rightarrow \mathsf{writeR}\ b; \mathsf{writeR}\ a; [\![q_0]\!] \\ \qquad\qquad \mid b \Rightarrow [\![q_1]\!] \\ \qquad\qquad \mid \$ \Rightarrow \mathsf{waitL}; \mathsf{writeR}\ b; \\ \qquad\qquad\qquad \mathsf{writeR}\ \$; \mathsf{closeR})\end{array}$$

This encoding of SFTs as proofs of type $\mathtt{Str}_{\hat{\Sigma}} \vdash \mathtt{Str}_{\hat{\Gamma}}$ is adequate at quite a fine-grained level – each SFT transition is matched by a proof reduction.

Theorem 4. *Let $T = (Q, \hat{\Sigma}, \hat{\Gamma}, \delta, \sigma, q_0)$ be a normal-form SFT. For all $q \in Q^{\mathsf{r}}$, if $\Delta \vdash (\mathsf{writeR}\ a; P) : \mathtt{Str}_{\hat{\Sigma}}$ and $\delta(a, q) = q_a$, then $(\mathsf{writeR}\ a; P) \triangleright [\![q]\!] \longrightarrow P \triangleright [\![q_a]\!]$.*

Proof. By straightforward calculation.

Corollary 1. *Let $T = (Q, \hat{\Sigma}, \hat{\Gamma}, \delta, \sigma, q_0)$ be a normal-form SFT. For all $w \in \hat{\Sigma}^+$ and $v \in \hat{\Gamma}^+$, if $w^{\mathcal{R}}\, q_0 \longrightarrow^* v^{\mathcal{R}}$, then $[\![w]\!] \triangleright [\![q_0]\!] \longrightarrow^* [\![v]\!]$.*

With SFTs encoded as cut-free proofs, SFT chains can easily be encoded as fixed-cut proofs – simply use the CUT rule to compose the encodings. For example, an SFT chain $(T_i)_{i=1}^n$ is encoded as $[\![q_1]\!]_{T_1} \triangleright \cdots \triangleright [\![q_n]\!]_{T_n}$. Because these occurrences of CUT do not occur inside any cycle, the encoding of an SFT chain is a fixed-cut proof.

4.4 Completing the Isomorphism: From Cut-Free Proofs to SFTs

In this section, we show that an SFT can be extracted from a cut-free proof of $\mathtt{Str}_{\hat{\Sigma}} \vdash_{\Theta} \mathtt{Str}_{\hat{\Gamma}}$, thereby completing the isomorphism.

We begin by inserting definitions in signature Θ so that each definition of type $\mathtt{Str}_{\hat{\Sigma}} \vdash \mathtt{Str}_{\hat{\Gamma}}$ has one of the forms

$$\begin{array}{lll} X = \mathsf{readL}_{a \in \hat{\Sigma}}(a \Rightarrow P_a) & \text{where } P_a = X_a & \text{if } a \in \Sigma_{\mathsf{i}} \\ & \text{and } P_e = \mathsf{waitL}; Y & \text{if } e \in \Sigma_{\mathsf{e}} \\ X = \mathsf{writeR}\ b; X_b & & \text{if } b \in \Gamma_{\mathsf{i}} \\ X = \mathsf{writeR}\ e; Z & & \text{if } e \in \Gamma_{\mathsf{e}} \end{array}$$

By inserting definitions we also put each Y of type $\cdot \vdash \mathtt{Str}_{\hat{\Gamma}}$ and each Z of type $\mathtt{Str}_{\hat{\Sigma}} \vdash \mathbf{1}$ into one of the forms

$$\begin{array}{lll} Y = \mathsf{writeR}\ b; Y_b & & \text{if } b \in \Gamma_{\mathsf{i}} \\ Y = \mathsf{writeR}\ e; W & & \text{if } e \in \Gamma_{\mathsf{e}} \\ Z = \mathsf{readL}_{a \in \hat{\Sigma}}(a \Rightarrow Q_a) & \text{where } Q_a = Z_a & \text{if } a \in \Sigma_{\mathsf{i}} \\ & \text{and } Q_e = \mathsf{waitL}; W & \text{if } e \in \Sigma_{\mathsf{e}} \end{array}$$

where definitions W of type $\cdot \vdash \mathbf{1}$ have the form $W = \mathsf{closeR}$. All of these forms are forced by the types, except in one case: P_e above has type $\mathbf{1} \vdash \mathtt{Str}_{\hat{\Gamma}}$, which

does not immediately force P_e to have the form $\mathsf{waitL}; Y$. However, by inversion on the type $\mathbf{1} \vdash \mathtt{Str}_{\hat{\Gamma}}$, we know that P_e is equivalent to a proof of the form $\mathsf{waitL}; Y$, *up to* commuting the $\mathbf{1}$L rule to the front.

From definitions in the above form, we can read off a normal-form SFT. Each variable becomes a state in the SFT. The normal-form conditions are manifest from the structure of the definitions: no read definition is reachable once an endmarker is read; no write definition is reachable once an endmarker is written; and a halt definition is reachable only by passing through a write of an endmarker.

Thus, cut-free proofs (up to $\mathbf{1}$L commuting conversion) are isomorphic to normal-form SFTs. Fixed-cut proofs are also then isomorphic to SFT chains by directly making the correspondence of fixed-cuts with chain links between neighboring SFTs.

5 SFT Composition by Cut Elimination

Subsequential functions enjoy closure under composition. This property is traditionally established by a direct SFT construction [14]. Having seen that SFTs are isomorphic to proofs of type $\mathtt{Str}_{\hat{\Sigma}} \vdash \mathtt{Str}_{\hat{\Gamma}}$, it's natural to wonder how this construction fits into this pleasing proof-theoretic picture. In this section, we show that, perhaps surprisingly, closure of SFTs under composition can indeed be explained proof-theoretically in terms of cut elimination.

5.1 Closure of SFTs under Composition

Composing two SFTs $T_1 = (Q_1, \hat{\Sigma}, \hat{\Gamma}, \delta_1, \sigma_1, q_1)$ and $T_2 = (Q_2, \hat{\Gamma}, \hat{\Omega}, \delta_2, \sigma_2, q_2)$ is simple: just compose their encodings. Because $[\![q_1]\!]_{T_1}$ and $[\![q_2]\!]_{T_2}$ have types $\mathtt{Str}_{\hat{\Sigma}} \vdash \mathtt{Str}_{\hat{\Gamma}}$ and $\mathtt{Str}_{\hat{\Gamma}} \vdash \mathtt{Str}_{\hat{\Omega}}$, respectively, the composition is $[\![q_1]\!]_{T_1} \triangleright [\![q_2]\!]_{T_2}$ and is well-typed.

By using an asynchronous, concurrent semantics of proof reduction [7], parallelism in the SFT chain can be exploited. For example, in the transducer chain $[\![w]\!] \triangleright [\![q_1]\!]_{T_1} \triangleright [\![q_2]\!]_{T_2} \triangleright [\![q_3]\!]_{T_3} \triangleright \cdots \triangleright [\![q_n]\!]_{T_n}$, the encoding of T_1 then react to the next symbol of input while T_2 is still absorbing T_1's first round of output.

Simply composing the encodings as the proof $[\![q_1]\!]_{T_1} \triangleright [\![q_2]\!]_{T_2}$ is suitable and very natural. But knowing that subsequential functions are closed under composition, what if we want to construct a single SFT that captures the same function as the composition?

The proof $[\![q_1]\!]_{T_1} \triangleright [\![q_2]\!]_{T_2}$ is a fixed-cut proof of $\mathtt{Str}_{\hat{\Sigma}} \vdash \mathtt{Str}_{\hat{\Omega}}$ because $[\![q_1]\!]_{T_1}$ and $[\![q_2]\!]_{T_2}$ are cut-free. Therefore, we know from Sects. 4.3 and 4.4 that, when applied to this composition, cut elimination will terminate with a cut-free circular proof of $\mathtt{Str}_{\hat{\Sigma}} \vdash \mathtt{Str}_{\hat{\Omega}}$. Because such proofs are isomorphic to SFTs, cut elimination constructs an SFT for the composition of T_1 and T_2. What is interesting, and somewhat surprising, is that a generic logical procedure such as cut elimination suffices for this construction – no extralogical design is necessary!

In fact, cut elimination yields the very same SFT that is traditionally used (see [14]) to realize the composition. We omit those details here.

5.2 DFA Closure under Complement and Inverse Homomorphism

Recall from Sect. 3.3 that our definition of SFTs subsumes deterministic finite automata (DFAs); an SFT that uses an endmarked output alphabet of $\hat{\Gamma} = (\{\}, \{a, r\})$ is a DFA that indicates acceptance or rejection of the input by producing a or r as its output.

Closure of SFTs under composition therefore implies closure of DFAs under complement and inverse homomorphism: For complement, compose the SFT-encoding of a DFA with an SFT over $\hat{\Gamma}$, *not*, that flips endmarkers. For inverse homomorphism, compose an SFT that captures homomorphism φ with the SFT-encoding of a DFA; the result recognizes $\varphi^{-1}(L) = \{w \mid \varphi(w) \in L\}$ where L is the language recognized by the DFA. (For endmarked strings, a homomorphism φ maps internal symbols to strings and endmarkers to endmarkers.) Thus, we also have cut elimination as a proof-theoretic explanation for the closure of DFAs under complement and inverse homomorphism.

6 Linear Communicating Automata

In the previous sections, we have established an isomorphism between the cut-free proofs of subsingleton logic and subsequential finite-state string transducers. We have so far been careful to avoid mixing circular proofs and general applications of the CUT rule. The reason is that cut elimination in general results in an infinite, but not necessarily circular, proof [9]. Unless the proof is circular, we can make no connection to machines with a finite number of states.

In this section, we consider the effects of incorporating the CUT in its full generality. We show that if we also relax conditions on circular proofs so that μ is a general – not least – fixed point, then proofs have the power of Turing machines. The natural computational interpretation of subsingleton logic with cuts is that of a typed form of communicating automata arranged with a linear network topology; these automata generalize Turing machines in two ways – the ability to insert and delete cells from the tape and the ability to spawn multiple machine heads that operate concurrently.

6.1 A Model of Linear Communicating Automata

First, we present a model of communicating automata arranged with a linear network topology. A *linear communicating automaton (LCA)* is an 8-tuple $M = (Q, \Sigma, \delta^{\mathsf{rL}}, \delta^{\mathsf{rR}}, \sigma^{\mathsf{wL}}, \sigma^{\mathsf{wR}}, \rho, q_0)$ where:

- Q is a finite set of states that is partitioned into (possibly empty) sets of left- and right-reading states, Q^{rL} and Q^{rR}; left- and right-writing states, Q^{wL} and Q^{wR}; spawn states, Q^{s}; and halt states, Q^{h};
- Σ is a finite alphabet;
- $\delta^{\mathsf{rL}} \colon \Sigma \times Q^{\mathsf{rL}} \to Q$ is a total function on left-reading states;
- $\delta^{\mathsf{rR}} \colon Q^{\mathsf{rR}} \times \Sigma \to Q$ is a total function on right-reading states;
- $\sigma^{\mathsf{wL}} \colon Q^{\mathsf{wL}} \to \Sigma \times Q$ is a total function on left-writing states;

– $\sigma^{\mathsf{wR}}\colon Q^{\mathsf{wR}} \to Q \times \Sigma$ is a total function on right-writing states;
– $\rho\colon Q^{\mathsf{s}} \to Q \times Q$ is a total function on spawn states;
– $q_0 \in Q$ is the initial state.

Configurations of the LCA M are strings w and v drawn from the set $(\Sigma^* Q)^* \Sigma^*$. Let $\longrightarrow$ be the least binary relation on configurations that satisfies the following.

$$
\begin{array}{lrll}
\textsc{read-l} & wa\,q\,v \longrightarrow w\,q_a\,v & \text{if } q \in Q^{\mathsf{rL}} \text{ and } \delta^{\mathsf{L}}(a,q) = q_a \\
\textsc{read-r} & w\,q\,bv \longrightarrow w\,q_b\,v & \text{if } q \in Q^{\mathsf{rR}} \text{ and } \delta^{\mathsf{R}}(q,b) = q_b \\
\textsc{write-l} & w\,q\,v \longrightarrow wa\,q_a\,v & \text{if } q \in Q^{\mathsf{wL}} \text{ and } \sigma^{\mathsf{L}}(q) = (a, q_a) \\
\textsc{write-r} & w\,q\,v \longrightarrow w\,q_b\,bv & \text{if } q \in Q^{\mathsf{wR}} \text{ and } \sigma^{\mathsf{R}}(q) = (q_b, b) \\
\textsc{spawn} & w\,q\,v \longrightarrow w\,q'\,q''\,v & \text{if } q \in Q^{\mathsf{s}} \text{ and } \rho(q) = (q', q'') \\
\textsc{halt} & w\,q\,v \longrightarrow wv & \text{if } q \in Q^{\mathsf{h}}
\end{array}
$$

The LCA M is said to *produce output* $v \in \Sigma^*$ *from input* $w \in \Sigma^*$ if there exists a sequence of configurations $u_0, \ldots, u_n$ such that *(i)* $u_0 = w^{\mathcal{R}}\,q_0$; *(ii)* $u_i \longrightarrow u_{i+1}$ for all $0 \le i < n$; and *(iii)* $u_n = v^{\mathcal{R}}$.

Notice that LCAs can certainly deadlock: a read state may wait indefinitely for the next symbol to arrive. LCAs also may exhibit races: two neighboring read states may compete to read the same symbol.

6.2 Comparing LCAs and Turing Machines

This model of LCAs makes their connections to Turing machines apparent. Each state q in the configuration represents a read/write head. Unlike Turing machines, LCAs may create and destroy tape cells as primitive operations (READ and WRITE rules) and create new heads that operate concurrently (SPAWN rule). In addition, LCAs are Turing complete.

Turing Machines. A *Turing machine* is a 4-tuple $M = (Q, \Sigma, \delta, q_0)$ where Q is a finite set of states that is partitioned into (possibly empty) sets of editing states, Q^{e} and halting states, Q^{h}; Σ is a finite alphabet; $\delta\colon (\Sigma \cup \{\epsilon\}) \times Q^{\mathsf{e}} \to Q \times \Sigma \times \{\mathsf{L}, \mathsf{R}\}$ is a function for editing states; and $q_0 \in Q$ is the initial state.

Configurations of the Turing machine M have one of two forms – either *(i)* $w\,q\,v$, where $w, v \in \Sigma^*$ and $q \in Q$; or *(ii)* w, where $w \in \Sigma^*$. In other words, the set of configurations is $\Sigma^* Q \Sigma^* \cup \Sigma^*$. Let $\longrightarrow$ be the least binary relation on configurations that satisfies the following conditions.

$$
\begin{array}{lrll}
\textsc{edit-l} & wa\,q\,v \longrightarrow w\,q_a\,bv & \text{if } \delta(a,q) = (q_a, b, \mathsf{L}) \\
 & q\,v \longrightarrow q_\epsilon\,bv & \text{if } \delta(\epsilon,q) = (q_\epsilon, b, \mathsf{L}) \\
\textsc{edit-r} & wa\,q\,cv \longrightarrow wbc\,q_a\,v & \text{if } \delta(a,q) = (q_a, b, \mathsf{R}) \\
 & wa\,q \longrightarrow wb\,q_a & \text{if } \delta(a,q) = (q_a, b, \mathsf{R}) \\
 & q\,cv \longrightarrow bc\,q_\epsilon\,v & \text{if } \delta(\epsilon,q) = (q_\epsilon, b, \mathsf{R}) \\
 & q \longrightarrow b\,q_\epsilon & \text{if } \delta(\epsilon,q) = (q_\epsilon, b, \mathsf{R}) \\
\textsc{halt} & w\,q\,v \longrightarrow wv & \text{if } q \in Q^{\mathsf{h}}
\end{array}
$$

LCAs are Turing Complete. A Turing machine can be simulated in a relatively straightforward way. First, we augment the alphabet with \$ and $\hat{}$ symbols as endmarkers. Each configuration $w\,q\,v$ becomes an LCA configuration $\$w\,q\,v\hat{}$. Each editing state q becomes a left-reading state in the encoding, and each halting state q becomes a halting state. If q is an editing state, then for each $a \in \Sigma$:

- If $\delta(a, q) = (q_a, b, \mathsf{L})$, introduce a fresh right-writing state q_b and let $\delta^{\mathsf{L}}(a, q) = q_b$ and $\sigma^{\mathsf{R}}(q_b) = (q_a, b)$. In this case, the first EDIT-L rule is simulated by $\$wa\,q\,v\hat{} \longrightarrow \$w\,q_b\,v\hat{} \longrightarrow \$w\,q_a\,bv\hat{}$.
- If $\delta(a, q) = (q_a, b, \mathsf{R})$, introduce fresh left-writing states q_b and q_c for each $c \in \Sigma$, a fresh right-reading state q'_b, and a fresh right-writing state $q_{\hat{}}$. Set $\delta^{\mathsf{L}}(a, q) = q_b$ and $\sigma^{\mathsf{L}}(q_b) = (b, q'_b)$. Also, set $\delta^{\mathsf{R}}(q'_b, c) = q_c$ for each $c \in \Sigma$, and $\delta^{\mathsf{R}}(q'_b, \hat{}) = q_{\hat{}}$. Finally, set $\sigma^{\mathsf{L}}(q_c) = (c, q_a)$ for each $c \in \Sigma$, and set $\sigma^{\mathsf{R}}(q_{\hat{}}) = (q_a, \hat{})$. In this case, the first and second EDIT-L rule are simulated by $\$wa\,q\,cv\hat{} \longrightarrow \$w\,q_b\,cv\hat{} \longrightarrow \$wb\,q'_b\,cv\hat{} \longrightarrow \$wb\,q_c\,v\hat{} \longrightarrow \$wbc\,q_a\,v\hat{}$ and $\$wa\,q\,\hat{} \longrightarrow \$w\,q_b\,\hat{} \longrightarrow \$wb\,q'_b\,\hat{} \longrightarrow \$wb\,q_{\hat{}} \longrightarrow \$wb\,q_a\,\hat{}$.
- The other cases are similar, so we omit them.

7 Extending $\oplus$,1,μ-Subsingleton Logic

In this section, we explore what happens when the CUT rule is allowed to occur along cycles in circular proofs. But first we extend $\oplus$,$\mathbf{1}$,μ-subsingleton logic and its computational interpretation with two other connectives: $\&$ and $\bot$.

7.1 Including $\&$ and $\bot$ in Subsingleton Logic

Figure 4 presents an extension of $\oplus$,$\mathbf{1}$,μ-subsingleton logic with $\&$ and $\bot$.

Once again, it will be convenient to generalize binary additive conjunctions to their n-ary, labeled form: $\&_{\ell \in L}\{\ell{:}A_\ell\}$ where L is nonempty. Contexts Δ still consist of exactly zero or one proposition, but conclusions γ may now be either empty or a single proposition.

The inference rules for $\&$ and $\bot$ are dual to those that we had for $\oplus$ and $\mathbf{1}$; once again, the inference rules become typing rules for proof terms. The $\&$R rule types a read operation, $\mathsf{readR}_{\ell \in L}(\ell \Rightarrow P_\ell)$, that branches on the label that was read; the label is read from the right-hand neighbor. Dually, the $\&$L rule types a write operation, $\mathsf{writeL}\ k; Q$, that emits label k to the left. The $\bot$R rule types an operation, $\mathsf{waitR}; P$, that waits for the right-hand neighbor to end; the $\bot$L rule types an operation, closeL, that signals to the left-hand neighbor. Finally, we restore ID as an inference rule, which types $\leftrightarrow$ as a forwarding operation.

Computational Interpretation: Well-Behaved LCAs. Already, the syntax of our proof terms suggests a computational interpretation of subsingleton logic with general cuts: well-behaved linear communicating automata.

The readL and readR operations, whose principal cut reductions read and consume a symbol from the left- and right-hand neighbors, respectively, become

$$
\begin{array}{rl}
\text{Types} & A, B, C ::= \cdots \mid \&_{\ell \in L}\{\ell{:}A_\ell\} \mid \bot \\
\text{Contexts} & \Delta ::= \cdot \mid A \\
\text{Conclusions} & \gamma ::= C \mid \cdot \\
\text{Proof terms} & P, Q ::= \cdots \mid \leftrightarrow \\
& \qquad \mid \mathsf{readR}_{\ell \in L}(\ell \Rightarrow P_\ell) \mid \mathsf{writeL}\ k; Q \\
& \qquad \mid \mathsf{waitR}; P \mid \mathsf{closeL} \\
\text{Signatures} & \Theta ::= \cdots
\end{array}
$$

$$
\frac{}{A \vdash_\Theta \leftrightarrow : A}\ \textsc{id}
$$

$$
\frac{\forall \ell \in L\colon\ \Delta \vdash_\Theta P_\ell : A_\ell}{\Delta \vdash_\Theta \mathsf{readR}_{\ell \in L}(\ell \Rightarrow P_\ell) : \&_{\ell \in L}\{\ell{:}A_\ell\}}\ \&\text{R} \qquad \frac{A_k \vdash_\Theta Q : \gamma \quad (k \in L)}{\&_{\ell \in L}\{\ell{:}A_\ell\} \vdash_\Theta \mathsf{writeL}\ k; Q : \gamma}\ \&\text{L}
$$

$$
\frac{\Delta \vdash_\Theta P : \cdot}{\Delta \vdash_\Theta \mathsf{waitR}; P : \bot}\ \mathbf{1}\text{R} \qquad \frac{}{\bot \vdash_\Theta \mathsf{closeL} : \cdot}\ \bot\text{L}
$$

$$
\begin{array}{c}
\leftrightarrow \triangleright Q \longrightarrow Q \qquad\qquad P \triangleright \leftrightarrow \longrightarrow P \\
\mathsf{readR}_{\ell \in L}(\ell \Rightarrow P_\ell) \triangleright (\mathsf{writeL}\ k; Q) \longrightarrow P_k \triangleright Q \\
(\mathsf{waitR}; P) \triangleright \mathsf{closeL} \longrightarrow P
\end{array}
$$

Fig. 4. A proof term assignment and principal cut reductions for the subsingleton sequent calculus when extended with & and $\bot$

left- and right-reading states. Similarly, the writeL and writeR operations that write a symbol to their left- and right-hand neighbors, respectively, become left- and right-writing states. Cuts, represented by the $\triangleright$ operation which creates a new read/write head, become spawning states. The ID rule, represented by the $\leftrightarrow$ operation, becomes a halting state.

Just as for SFTs, this interpretation is adequate at a quite fine-grained level in that LCA transitions are matched by proof reductions. Moreover, the types in our interpretation of subsingleton logic ensure that the corresponding LCA is well-behaved. For example, the corresponding LCAs cannot deadlock because cut elimination can always make progress, as proved by Fortier and Santocanale [9]; those LCAs also do not have races in which two neighboring heads compete to read the same symbol because readR and readL have different types and therefore cannot be neighbors. Due to space constraints, we omit a discussion of the details.

7.2 Subsingleton Logic Is Turing Complete

Once we allow general occurrences of CUT, we can in fact simulate Turing machines and show that subsingleton logic is Turing complete. For each state q in the Turing machine, define an encoding $[\![q]\!]$ as follows.

If q is an editing state, let $[\![q]\!] = \mathsf{readL}_{a\in\Sigma}(a \Rightarrow P_{q,a} \mid \$ \Rightarrow P'_q)$ where

$$P_{q,a} = \begin{cases} [\![q_a]\!] \triangleright (\mathsf{writeL}\ b; \leftrightarrow) & \text{if } \delta(a,q) = (q_a, b, \mathsf{L}) \\ \mathsf{readR}_{c\in\Sigma}(c \Rightarrow (\mathsf{writeR}\ c; \mathsf{writeR}\ b; \leftrightarrow) \triangleright [\![q_a]\!] \\ \qquad \mid \hat{} \Rightarrow (\mathsf{writeR}\ b; \leftrightarrow) \triangleright [\![q_a]\!]) \\ \qquad\qquad \triangleright (\mathsf{writeL}\ \hat{};\leftrightarrow) & \text{if } \delta(a,q) = (q_a, b, \mathsf{R}) \end{cases}$$

and

$$P'_q = \begin{cases} (\mathsf{writeR}\ \$; \leftrightarrow) \triangleright [\![q_\epsilon]\!] \triangleright (\mathsf{writeL}\ b; \leftrightarrow) & \text{if } \delta(\epsilon,q) = (q_\epsilon, b, \mathsf{L}) \\ \mathsf{readR}_{c\in\Sigma}(c \Rightarrow (\mathsf{writeR}\ c; \mathsf{writeR}\ b; \leftrightarrow) \triangleright [\![q_\epsilon]\!] \\ \qquad \mid \hat{} \Rightarrow (\mathsf{writeR}\ b; \leftrightarrow) \triangleright [\![q_\epsilon]\!] \triangleright (\mathsf{writeL}\ \hat{};\leftrightarrow)) & \text{if } \delta(\epsilon,q) = (q_\epsilon, b, \mathsf{R}) \end{cases}$$

If q is a halt state, let $[\![q]\!] = \mathsf{readR}_{c\in\Sigma}(c \Rightarrow (\mathsf{writeR}\ c; \leftrightarrow) \triangleright [\![q]\!] \mid \hat{} \Rightarrow \leftrightarrow)$. Surprisingly, these definitions $[\![q]\!]$ are in fact well-typed at $\mathtt{Tape} \vdash \mathtt{epaT}$, where

$$\mathtt{Tape} = \mu\alpha.\, \oplus_{a\in\Sigma}\{a{:}\alpha, \${:}\mathbf{1}\}$$
$$\mathtt{epaT} = \mu\alpha.\, \&_{a\in\Sigma}\{a{:}\alpha, \hat{}{:}\mathtt{Tape}\}\,.$$

This means that Turing machines cannot get stuck!

Of course, Turing machines may very well loop indefinitely. And so, for the above circular proof terms to be well-typed, we must give up on μ being an *inductive* type and relax μ to be a *general recursive* type. This amounts to dropping the requirement that every cycle in a circular proof is a left μ-trace.

It is also possible to simulate Turing machines in a well-typed way without using $\&$. Occurrences of $\&$, readR, and writeL are removed by instead using $\oplus$ and its constructs in a continuation-passing style. This means that Turing completeness depends on the interaction of general cuts and general recursion, not on any subtleties of interaction between $\oplus$ and $\&$.

8 Conclusion

We have taken the computational interpretation of linear logic first proposed by Caires et al. [3] and restricted it to a fragment with just $\oplus$ and $\mathbf{1}$, but added least fixed points and circular proofs [9]. Cut-free proofs in this fragment are in an elegant Curry-Howard correspondence with subsequential finite state transducers. Closure under composition, complement, inverse homomorphism, intersection and union can then be realized uniformly by cut elimination. We plan to investigate if closure under concatenation and Kleene star, usually proved via a detour through nondeterministic automata, can be similarly derived.

When we allow arbitrary cuts, we obtain linear communicating automata, which is a Turing-complete class of machines. Some preliminary investigation leads us to the conjecture that we can also obtain deterministic pushdown automata as a naturally defined logical fragment. Conversely, we can ask if the restrictions of the logic to least or greatest fixed points, that is, inductive or coinductive types with corresponding restrictions on the structure of circular proofs yields interesting or known classes of automata.

Our work on communicating automata remains significantly less general than Deniélou and Yoshida's analysis using multiparty session types [6]. Instead of multiparty session types, we use only a small fragment of binary session types; instead of rich networks of automata, we limit ourselves to finite chains of machines. And in our work, machines can terminate and spawn new machines, and both operational and typing aspects of LCAs arise naturally from logical origins.

Finally, in future work we would like to explore if we can design a *subsingleton type theory* and use it to reason intrinsically about properties of automata.

References

1. Baelde, D.: Least and greatest fixed points in linear logic. ACM Transactions on Computational Logic 13(1) (2012)
2. Baelde, D., Doumane, A., Saurin, A.: Infinitary proof theory: The multiplicative additive case. In: 25th Conference on Computer Science Logic. LIPIcs, vol. 62, pp. 42:1–42:17 (2016)
3. Caires, L., Pfenning, F.: Session types as intuitionistic linear propositions. In: 21st International Conference on Concurrency Theory. LNCS, vol. 6269, pp. 222–236 (2010)
4. Church, A., Rosser, J.: Some properties of conversion. Transactions of the American Mathematical Society 39(3), 472–482 (May 1936)
5. Curry, H.B.: Functionality in combinatory logic. Proceedings of the National Academy of Sciences, U.S.A. 20, 584–590 (1934)
6. Deniélou, P., Yoshida, N.: Multiparty session types meet communicating automata. In: Seidl, H. (ed.) Proceedings of the 21st European Symposium on Programming (ESOP). pp. 194–213. Springer LNCS 7211 (Mar 2012)
7. DeYoung, H., Caires, L., Pfenning, F., Toninho, B.: Cut reduction in linear logic as asynchronous session-typed communication. In: 21st Conference on Computer Science Logic. LIPIcs, vol. 16, pp. 228–242 (2012)
8. Dummett, M.: The Logical Basis of Metaphysics. Harvard University Press (1991), from the William James Lectures, 1976.
9. Fortier, J., Santocanale, L.: Cuts for circular proofs: Semantics and cut elimination. In: 22nd Conference on Computer Science Logic. LIPIcs, vol. 23, pp. 248–262 (2013)
10. Gay, S., Hole, M.: Subtyping for session types in the pi calculus. Acta Informatica 42(2), 191–225 (2005)
11. Girard, J.Y.: Linear logic. Theoretical Computer Science 50(1), 1–102 (1987)
12. Howard, W.A.: The formulae-as-types notion of construction (1969), unpublished note. An annotated version appeared in: *To H.B. Curry: Essays on Combinatory Logic, Lambda Calculus and Formalism*, 479–490, Academic Press (1980)
13. Martin-Löf, P.: On the meanings of the logical constants and the justifications of the logical laws. Nordic Journal of Philosophical Logic 1(1), 11–60 (1996)
14. Mohri, M.: Finite-state transducers in language and speech processing. Journal of Computational Linguistics 23(2), 269–311 (1997)
15. Schützenberger, M.P.: Sur une variante des fonctions sequentielles. Theoretical Computer Science 4(1), 47–57 (1977)
16. Turing, A.M.: On computable numbers, with an application to the Entscheidungsproblem. Proceedings of the London Mathematical Society 42(2), 230–265 (1937)