Substructural Proofs as Automata

Frank Pfenning

Department of Computer Science
Carnegie Mellon University

14th Asian Symposium on
Programming Languages and Systems (APLAS 2016)
Invited Talk
Hanoi, Vietnam
November 22, 2016

Church and Turing

Computation		
λ-Calculus [Church 1936]		
Turing Machines [Turing 1937]		

Church and Turing

Computation	Logic	Synthesis
λ-Calculus [Church 1936]	Intuitionistic Logic [Heyting 1930]	Proofs as Programs [Howard 1969]
Turing Machines [Turing 1937]		

Church and Turing

Computation	**Logic**	**Synthesis**
λ-Calculus [Church 1936]	Intuitionistic Logic [Heyting 1930]	Proofs as Programs [Howard 1969]
Turing Machines [Turing 1937]	?	?

Church and Turing

Computation	Logic	Synthesis
λ-Calculus [Church 1936]	Intuitionistic Logic [Heyting 1930]	Proofs as Programs [Howard 1969]
Turing Machines [Turing 1937]	?	?
	Subsingleton Logic [Santocanale 2001]	

Church and Turing

Computation	Logic	Synthesis
λ-Calculus [Church 1936]	Intuitionistic Logic [Heyting 1930]	Proofs as Programs [Howard 1969]
Turing Machines [Turing 1937]	?	?
Linear Communicating Automata	Subsingleton Logic [Santocanale 2001]	Substructural Proofs as Automata [this paper]

Church and Turing

Computation	**Logic**	**Synthesis**
λ-Calculus [Church 1936]	Intuitionistic Logic [Heyting 1930]	Proofs as Programs [Howard 1969]
Turing Machines [Turing 1937]	?	?
Linear Communicating Automata	Subsingleton Logic [Santocanale 2001]	Substructural Proofs as Automata [this paper]
Subsequential Finite State Transducers	Fixed Cut Subsingleton Logic	[this paper]

Curry-Howard Correspondence

- For a constructive logic, relate:

Logic	**Computation**
Proposition	Type
Proof	Program
Proof Reduction	Computation

- Design of a language and logic for reasoning about its programs go hand in hand
- Full synthesis takes place in type theory
- Considerable ingenuity may be required
- Best case: an isomorphism

Examples of Isomorphisms

Logic	Computation
Intuitionistic axiomatic proofs	Combinatory reduction [Curry 1935]
Intuitionistic natural deduction	Functional computation [Howard 1969]
Temporal logic	Partial evaluation [Davies 1996]
S4 modal logic	Staged computation [Davies & Pf 1996]
Linear sequent calculus	Concurrent computation [Caires & Pf 2010] [Wadler 2012]
Fixed cut subsingleton logic	Finite state transduction [this paper]

Overview

1. Subsingleton logic
2. Proof reduction semantics
3. Representing strings
4. From transducers to proofs
5. From proofs to transducers
6. Two applications
7. Full subsingleton logic
8. Encoding Turing machines
9. Linear communicating automata

Subsingleton Logic

- Fragment of linear logic with 0 or 1 antecedents

$$\begin{array}{lcl} A, B, C & ::= & A \oplus B \mid \mathbf{1} \mid A \mathbin{\&} B \mid \bot \\ \Delta & ::= & \cdot \mid A \end{array}$$

- Rules for the $\oplus, \mathbf{1}$-fragment

$$\dfrac{}{A \vdash A}\ \mathrm{id}_A \qquad \dfrac{\Delta \vdash B \quad B \vdash C}{\Delta \vdash C}\ \mathrm{cut}_B$$

$$\dfrac{\Delta \vdash A}{\Delta \vdash A \oplus B}\ \oplus R_1 \quad \dfrac{\Delta \vdash B}{\Delta \vdash A \oplus B}\ \oplus R_2 \qquad \dfrac{A \vdash C \quad B \vdash C}{A \oplus B \vdash C}\ \oplus L$$

$$\dfrac{}{\cdot \vdash \mathbf{1}}\ \mathbf{1}R \qquad \dfrac{\cdot \vdash C}{\mathbf{1} \vdash C}\ \mathbf{1}L$$

A Computational Interpretation

- Judgment $\Delta \vdash P : A$
- Δ and A are the left and right interface for process P
- Cut as (non-commutative!) parallel composition

$$\frac{\Delta \vdash P : A \quad A \vdash Q : C}{\Delta \vdash (P \mid Q) : C}\ \mathrm{cut}_A$$

- Identity as forwarding

$$\frac{}{A \vdash \leftrightarrow : A}\ \mathrm{id}_A$$

Process Configurations

- A process configuration Ω is an ordered parallel composition of processes with matching interface types

$$\Delta \vdash P_1 \mid P_2 \mid \ldots \mid P_n : A_n$$

- Computation for cut and identity

$$\begin{array}{llllll}
\text{cut} & : & \Omega_L \mid (P \mid Q) \mid \Omega_R & \longrightarrow & \Omega_L \mid P \mid Q \mid \Omega_R \\
\text{id} & : & \Omega_L \mid (\leftrightarrow) \mid \Omega_R & \longrightarrow & \Omega_L \mid \Omega_R
\end{array}$$

Process Configurations

- A process configuration Ω is an ordered parallel composition of processes with matching interface types

$$\Delta \vdash P_1 \mid P_2 \mid \ldots \mid P_n : A_n$$

- Computation for cut and identity

$$\begin{array}{lclcl}
\mathsf{cut} & : & \Omega_L \mid_\Delta (P \mid_A Q) \mid_C \Omega_R & \longrightarrow & \Omega_L \mid_\Delta P \mid_A Q \mid_C \Omega_R \\
\mathsf{id} & : & \Omega_L \mid \ (\leftrightarrow) \mid \ \Omega_R & \longrightarrow & \Omega_L \mid \ \Omega_R
\end{array}$$

Process Configurations

- A process configuration Ω is an ordered parallel composition of processes with matching interface types

$$\Delta \vdash P_1 \mid P_2 \mid \ldots \mid P_n : A_n$$

- Computation for cut and identity

$$\begin{array}{lcllcl} \mathsf{cut} & : & \Omega_L \mid_\Delta (P \mid_A Q) \mid_C \Omega_R & \longrightarrow & \Omega_L \mid_\Delta P \mid_A Q \mid_C \Omega_R \\ \mathsf{id} & : & \Omega_L \mid_A (\leftrightarrow) \mid_A \Omega_R & \longrightarrow & \Omega_L \mid_A \Omega_R \end{array}$$

Cut Reduction as the Engine of Computation

- Consider when $\oplus R_i$ meets $\oplus L$
- $\oplus L$ is prepared for either A or B to be true
- $\oplus R_1$ selects A, $\oplus R_2$ selects B
- Reduce principal cut to smaller cuts

$$\dfrac{\dfrac{\Delta \vdash A}{\Delta \vdash A \oplus B}\,\oplus R_1 \qquad \dfrac{A \vdash C \quad B \vdash C}{A \oplus B \vdash C}\,\oplus L}{\Delta \vdash C}\,\mathsf{cut}_{A \oplus B}$$

Cut Reduction as the Engine of Computation

- Consider when $\oplus R_i$ meets $\oplus L$
- $\oplus L$ is prepared for either A or B to be true
- $\oplus R_1$ selects A, $\oplus R_2$ selects B
- Reduce principal cut to smaller cuts

$$\dfrac{\dfrac{\Delta \vdash A}{\Delta \vdash A \oplus B}\,\oplus R_1 \quad \dfrac{A \vdash C \quad B \vdash C}{A \oplus B \vdash C}\,\oplus L}{\Delta \vdash C}\,\mathsf{cut}_{A\oplus B}$$

$$\longrightarrow \quad \dfrac{\Delta \vdash A \quad A \vdash C}{\Delta \vdash C}\,\mathsf{cut}_A$$

- Plus symmetric version

Process Assignment and Reduction for $A \oplus B$

- $\oplus R_i$ send, $\oplus L$ receives

$$\frac{\Delta \vdash P : A}{\Delta \vdash \mathsf{R}.\pi_1 \ \mathbf{;} \ P : A \oplus B} \oplus R_1 \qquad \frac{\Delta \vdash P : B}{\Delta \vdash \mathsf{R}.\pi_2 \ \mathbf{;} \ P : A \oplus B} \oplus R_2$$

$$\frac{A \vdash Q_1 : C \qquad\qquad B \vdash Q_2 : C}{A \oplus B \vdash \mathsf{caseL}\ (\pi_1 \Rightarrow Q_1 \mid \pi_2 \Rightarrow Q_2) : C} \oplus L$$

Process Assignment and Reduction for $A \oplus B$

- $\oplus R_i$ send, $\oplus L$ receives

$$\frac{\Delta \vdash P : A}{\Delta \vdash \mathsf{R}.\pi_1 \; ; \; P : A \oplus B} \oplus R_1 \qquad \frac{\Delta \vdash P : B}{\Delta \vdash \mathsf{R}.\pi_2 \; ; \; P : A \oplus B} \oplus R_2$$

$$\frac{A \vdash Q_1 : C \qquad\qquad B \vdash Q_2 : C}{A \oplus B \vdash \mathsf{caseL}\,(\pi_1 \Rightarrow Q_1 \mid \pi_2 \Rightarrow Q_2) : C} \oplus L$$

- Computation rules (apply anywhere in a configuration)

$$\begin{array}{lclcl} (\mathsf{R}.\pi_1 \; ; \; P) \mid & \mathsf{caseL}\,(\pi_1 \Rightarrow Q_1 \mid \pi_2 \Rightarrow Q_2) & \longrightarrow & P \mid & Q_1 \\ (\mathsf{R}.\pi_2 \; ; \; P) \mid & \mathsf{caseL}\,(\pi_1 \Rightarrow Q_1 \mid \pi_2 \Rightarrow Q_2) & \longrightarrow & P \mid & Q_2 \end{array}$$

Process Assignment and Reduction for $A \oplus B$

- $\oplus R_i$ send, $\oplus L$ receives

$$\frac{\Delta \vdash P : A}{\Delta \vdash \mathsf{R}.\pi_1 \,;\, P : A \oplus B}\ \oplus R_1 \qquad \frac{\Delta \vdash P : B}{\Delta \vdash \mathsf{R}.\pi_2 \,;\, P : A \oplus B}\ \oplus R_2$$

$$\frac{A \vdash Q_1 : C \qquad\qquad B \vdash Q_2 : C}{A \oplus B \vdash \mathsf{caseL}\,(\pi_1 \Rightarrow Q_1 \mid \pi_2 \Rightarrow Q_2) : C}\ \oplus L$$

- Computation rules (apply anywhere in a configuration)

$$\begin{array}{lcl} (\mathsf{R}.\pi_1 \,;\, P) \mid_{A \oplus B} \mathsf{caseL}\,(\pi_1 \Rightarrow Q_1 \mid \pi_2 \Rightarrow Q_2) & \longrightarrow & P \mid_A Q_1 \\ (\mathsf{R}.\pi_2 \,;\, P) \mid_{A \oplus B} \mathsf{caseL}\,(\pi_1 \Rightarrow Q_1 \mid \pi_2 \Rightarrow Q_2) & \longrightarrow & P \mid_B Q_2 \end{array}$$

Process Assignment and Reduction for $\mathbf{1}$

- $\mathbf{1}R$ sends, $\mathbf{1}L$ receives

$$\frac{}{\cdot \vdash \mathsf{closeR} : \mathbf{1}}\ \mathbf{1}R \qquad\qquad \frac{\cdot \vdash Q : C}{\mathbf{1} \vdash \mathsf{waitL}\ ;\ Q : C}\ \mathbf{1}L$$

- Computation rule

$$(\mathsf{closeR}) \mid_{\mathbf{1}} (\mathsf{waitL}\ ;\ Q) \quad \longrightarrow \quad Q$$

Generalize to Labeled Sum

- In programming, need more than two branches

$$A \quad ::= \quad \oplus_{\ell \in L}\{\ell : A_\ell\} \mid \mathbf{1} \mid \&_{\ell \in L}\{\ell : A_\ell\} \mid \bot$$

- Generalize rules straightforwardly

$$\frac{\Delta \vdash P : A_k \quad (k \in L)}{\Delta \vdash \mathsf{R}.k\ \mathbf{;}\ P : \oplus_{\ell \in L}\{\ell : A_\ell\}} \oplus R_k$$

$$\frac{A_\ell \vdash Q_\ell : C \quad (\forall \ell \in L)}{\oplus_{\ell \in L}\{\ell : A_\ell\} \vdash \mathsf{caseL}\ (\ell \Rightarrow Q_\ell)_{\ell \in L} : C} \oplus L$$

- Computation rules (apply anywhere in a configuration)

$$(\mathsf{R}.k\ \mathbf{;}\ P) \mid \mathsf{caseL}\ (\ell \Rightarrow Q_\ell)_{\ell \in L} \quad \longrightarrow \quad P \mid Q_k$$

Summary of Process Reduction

- $P : \oplus_{\ell \in L}\{\ell : A_\ell\}$ sends $k \in L$, continues as A_k
- $P : \mathbf{1}$ sends closeR and terminates
- Computation rules (apply anywhere in a configuration)

$$
\begin{array}{lcl}
(P \mid Q) & \longrightarrow & P \mid Q \\
(\leftrightarrow) & \longrightarrow & \cdot \\
(\mathsf{R}.k \;\mathbf{;}\; P) \mid \mathsf{caseL}\,(\ell \Rightarrow Q_\ell)_{\ell \in L} & \longrightarrow & P \mid Q_k \\
(\mathsf{closeR}) \mid (\mathsf{waitL} \;\mathbf{;}\; Q) & \longrightarrow & Q
\end{array}
$$

- Configurations are ordered: no explicit channels needed for communication

Towards Automata: Representing Strings

- Symbols $a \in \Sigma$ as labels $a \in \Sigma$
- Strings as sequences of messages
- Finish with endmarker \$ and close

$$\ulcorner a_1\, a_2 \ldots a_n \urcorner = \mathsf{R}.a_1 \,\mathbf{;}\, \mathsf{R}.a_2 \,\mathbf{;}\, \ldots \,\mathbf{;}\, \mathsf{R}.a_n \,\mathbf{;}\, \mathsf{R}.\$ \,\mathbf{;}\, \mathsf{closeR}$$

- How do we type this?

Towards Automata: Representing Strings

- Symbols $a \in \Sigma$ as labels $a \in \Sigma$
- Strings as sequences of messages
- Finish with endmarker \$ and close

$$\ulcorner a_1\, a_2 \ldots a_n \urcorner = \mathsf{R}.a_1 \; \mathbf{;} \; \mathsf{R}.a_2 \; \mathbf{;} \; \ldots \; \mathbf{;} \; \mathsf{R}.a_n \; \mathbf{;} \; \mathsf{R}.\$ \; \mathbf{;} \; \mathsf{closeR}$$

- How do we type this?
- Need inductive type! For $\Sigma = \{a, b, \ldots\}$ we define

$$\mathsf{string}_\Sigma \quad = \quad \oplus\{a : ?, b : ?, \ldots, \$: ?\}$$

Towards Automata: Representing Strings

- Symbols $a \in \Sigma$ as labels $a \in \Sigma$
- Strings as sequences of messages
- Finish with endmarker \$ and close

$$\ulcorner a_1\, a_2 \ldots a_n \urcorner = \mathrm{R}.a_1 \;\mathbf{;}\; \mathrm{R}.a_2 \;\mathbf{;}\; \ldots \;\mathbf{;}\; \mathrm{R}.a_n \;\mathbf{;}\; \mathrm{R}.\$ \;\mathbf{;}\; \mathrm{closeR}$$

- How do we type this?
- Need inductive type! For $\Sigma = \{a, b, \ldots\}$ we define

$$\mathrm{string}_\Sigma \quad = \quad \oplus\{a : \mathrm{string}_\Sigma, b : \mathrm{string}_\Sigma, \ldots, \$: ?\}$$

Towards Automata: Representing Strings

- Symbols $a \in \Sigma$ as labels $a \in \Sigma$
- Strings as sequences of messages
- Finish with endmarker \$ and close

$$\ulcorner a_1\, a_2 \ldots a_n \urcorner = \mathrm{R}.a_1 \;\mathbf{;}\; \mathrm{R}.a_2 \;\mathbf{;}\; \ldots \;\mathbf{;}\; \mathrm{R}.a_n \;\mathbf{;}\; \mathrm{R}.\$ \;\mathbf{;}\; \mathrm{closeR}$$

- How do we type this?
- Need inductive type! For $\Sigma = \{a, b, \ldots\}$ we define

$$\mathrm{string}_\Sigma \quad = \quad \oplus\{a : \mathrm{string}_\Sigma, b : \mathrm{string}_\Sigma, \ldots, \$: \mathbf{1}\}$$

Towards Automata: Representing Strings

- Symbols $a \in \Sigma$ as labels $a \in \Sigma$
- Strings as sequences of messages
- Finish with endmarker \$ and close

$$\ulcorner a_1\, a_2 \ldots a_n \urcorner = \mathrm{R}.a_1 \mathbin{\textbf{;}} \mathrm{R}.a_2 \mathbin{\textbf{;}} \ldots \mathbin{\textbf{;}} \mathrm{R}.a_n \mathbin{\textbf{;}} \mathrm{R}.\$ \mathbin{\textbf{;}} \mathrm{closeR}$$

- How do we type this?
- Need inductive type! For $\Sigma = \{a, b, \ldots\}$ we define

$$\begin{array}{rcl} \mathrm{string}_\Sigma & = & \oplus\{a : \mathrm{string}_\Sigma, b : \mathrm{string}_\Sigma, \ldots, \$: \mathbf{1}\} \\ & = & \oplus_{a \in \Sigma}\{a : \mathrm{string}_\Sigma, \$: \mathbf{1}\} \end{array}$$

Towards Automata: Representing Strings

- Symbols $a \in \Sigma$ as labels $a \in \Sigma$
- Strings as sequences of messages
- Finish with endmarker \$ and close

$$\ulcorner a_1\, a_2 \ldots a_n \urcorner = \mathrm{R}.a_1 \,\mathbf{;}\, \mathrm{R}.a_2 \,\mathbf{;}\, \ldots \,\mathbf{;}\, \mathrm{R}.a_n \,\mathbf{;}\, \mathrm{R}.\$ \,\mathbf{;}\, \mathrm{closeR}$$

- How do we type this?
- Need inductive type! For $\Sigma = \{a, b, \ldots\}$ we define

$$\begin{aligned} \mathrm{string}_\Sigma &= \oplus\{a : \mathrm{string}_\Sigma, b : \mathrm{string}_\Sigma, \ldots, \$: \mathbf{1}\} \\ &= \oplus_{a \in \Sigma}\{a : \mathrm{string}_\Sigma, \$: \mathbf{1}\} \end{aligned}$$

- Sometimes omit the subscript Σ

A First Bijection

- Representing strings

$$\mathsf{string}_\Sigma = \oplus_{a\in\Sigma}\{a : \mathsf{string}_\Sigma, \$: \mathbf{1}\}$$

$$\ulcorner a_1\, a_2 \ldots a_n \urcorner = \mathsf{R}.a_1 \mathbin{;} \mathsf{R}.a_2 \mathbin{;} \ldots \mathbin{;} \mathsf{R}.a_n \mathbin{;} \mathsf{R}.\$ \mathbin{;} \mathsf{closeR}$$

- For a string w over alphabet Σ we have

$$\cdot \vdash \ulcorner w \urcorner : \mathsf{string}_\Sigma$$

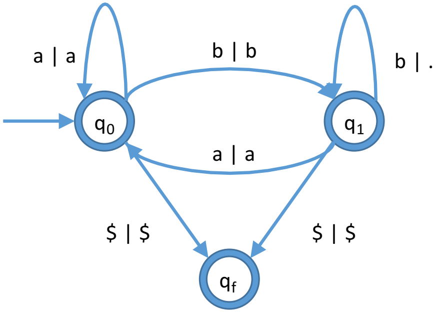
- For any cut-free proof P, if

$$\cdot \vdash P : \mathsf{string}_\Sigma$$

  then $P = \ulcorner w \urcorner$ for some string w over Σ
- There is a compositional bijection between strings and cut-free processes $P : \mathsf{string}$

Subsequential Finite State Transducers

- A subseqential finite state transducer (STM) starts in some initial state q_0 and
  1. reads one symbol from an input string
  2. writes zero or more symbols to an output string
  3. transitions to the next state
- Example: compressing each run of b's into one b



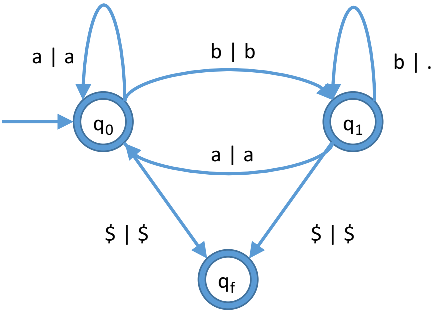
$$\overbrace{\$\, c_n \ldots c_1}^{\text{input}}\ q\ \overbrace{d_k \ldots d_1}^{\text{output}}$$

As mixed string rewriting

$$\begin{aligned}
a\, q_0 &\longrightarrow q_0\, a \\
b\, q_0 &\longrightarrow q_1\, b \\
\$\, q_0 &\longrightarrow \$ \\
a\, q_1 &\longrightarrow q_0\, a \\
b\, q_1 &\longrightarrow q_1 \\
\$\, q_1 &\longrightarrow \$
\end{aligned}$$

SFTs as Processes

$$
\begin{aligned}
Q_0 = \text{caseL } (& \, a \Rightarrow \text{R}.a \text{ ; } Q_0 \\
| & \, b \Rightarrow \text{R}.b \text{ ; } Q_1 \\
| & \, \$ \Rightarrow \text{R}.\$ \text{ ; waitL ; closeR}) \\
Q_1 = \text{caseL } (& \, a \Rightarrow \text{R}.a \text{ ; } Q_0 \\
| & \, b \Rightarrow Q_1 \\
| & \, \$ \Rightarrow \text{R}.\$ \text{ ; waitL ; closeR})
\end{aligned}
$$



As mixed string rewriting

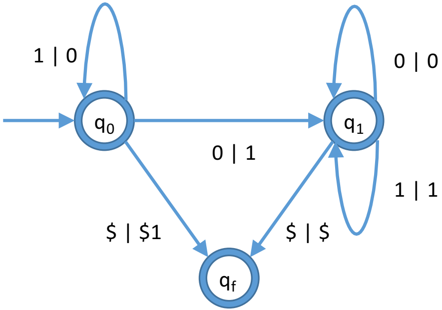
$$
\begin{array}{lcl}
a\, q_0 & \longrightarrow & q_0\, a \\
b\, q_0 & \longrightarrow & q_1\, b \\
\$\, q_0 & \longrightarrow & \$ \\
a\, q_1 & \longrightarrow & q_0\, a \\
b\, q_1 & \longrightarrow & q_1 \\
\$\, q_1 & \longrightarrow & \$
\end{array}
$$

Circular Proofs

- Requires circular (coinductive) proofs
  [Santocanale 2001] [Fortier & Santocanale 2013]
  [Baelde, Doumane, & Saurin 2016]
- For fixed cut proofs (no cycle contains a cut), cut elimination yields cut-free circular proofs
- With arbitrary cuts, elimination may yield infinite proofs
- Here: circular proofs as mutually recursive process defns
- Computation ($\sim$ cut elimination) will terminate if all process definitions are cut-free

SFTs Example 2: Incrementing a Bit String

- Example: Incrementing a bit string
- Least significant bit arrives first
- q_0 increments, q_1 copies

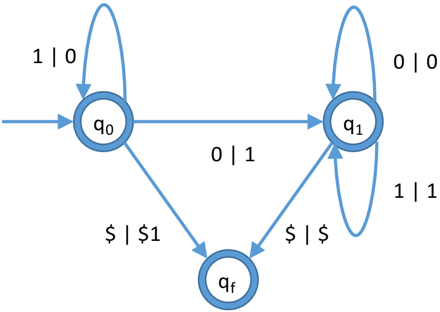


As mixed string rewriting

$$
\begin{array}{lcl}
0\,q_0 & \longrightarrow & q_1\,1 \\
1\,q_0 & \longrightarrow & q_0\,0 \\
\$\,q_0 & \longrightarrow & \$\,1 \\
0\,q_1 & \longrightarrow & q_1\,0 \\
1\,q_1 & \longrightarrow & q_1\,1 \\
\$\,q_1 & \longrightarrow & \$
\end{array}
$$

SFTs Example 2: Incrementing a Bit String

$$
\begin{aligned}
Q_0 = \mathsf{caseL}\ (\ & 0 \Rightarrow \mathsf{R}.1\ ;\ Q_1 \\
\mid\ & 1 \Rightarrow \mathsf{R}.0\ ;\ Q_0 \\
\mid\ & \$ \Rightarrow \mathsf{R}.1\ ;\ \mathsf{R}.\$\ ;\ \mathsf{waitL}\ ;\ \mathsf{closeR}) \\
Q_1 = \mathsf{caseL}\ (\ & 0 \Rightarrow \mathsf{R}.0\ ;\ Q_1 \\
\mid\ & 1 \Rightarrow \mathsf{R}.1\ ;\ Q_1 \\
\mid\ & \$ \Rightarrow \mathsf{R}.\$\ ;\ \mathsf{waitL}\ ;\ \mathsf{closeR})
\end{aligned}
$$



As mixed string rewriting

$$
\begin{array}{lcl}
0\ q_0 & \longrightarrow & q_1\ 1 \\
1\ q_0 & \longrightarrow & q_0\ 0 \\
\$\ q_0 & \longrightarrow & \$\ 1 \\
0\ q_1 & \longrightarrow & q_1\ 0 \\
1\ q_1 & \longrightarrow & q_1\ 1 \\
\$\ q_1 & \longrightarrow & \$
\end{array}
$$

A Second Bijection

Theorem (Representation of SFTs)

There is a bijection between SFTs T from Σ to Γ and cut-free, identity-free, circular processes P with

$$\mathsf{string}_\Sigma \vdash P : \mathsf{string}_\Gamma$$

such that

$$\$w^R\, q_o \longrightarrow^* \$v^R \quad \textit{iff} \quad \ulcorner w \urcorner \mid P \longrightarrow^* \ulcorner v \urcorner$$

with corresponding steps.

A Second Bijection

Theorem (Representation of SFTs)

There is a bijection between SFTs T from Σ to Γ and cut-free, identity-free, circular processes P with

$$\mathsf{string}_\Sigma \vdash P : \mathsf{string}_\Gamma$$

such that

$$\$ w^R\, q_o \longrightarrow^* \$ v^R \quad \textit{iff} \quad \ulcorner w \urcorner \mid P \longrightarrow^* \ulcorner v \urcorner$$

with corresponding steps.

- Technical condition on the operational semantics
  - Either use asynchronous message passing
  - or reduce under output prefixes (see paper)
  - or use an observer process to force computation

Processes as String Transducers

- Recall $\mathsf{string}_\Sigma = \oplus_{a\in\Sigma}\{a : \mathsf{string}_\Sigma, \$: \mathbf{1}\}$
- What can a cut-free, identity-free process P with $\mathsf{string}_\Sigma \vdash P : \mathsf{string}_\Gamma$ do?
  - Branch on a label received from the left
    - If it receives $a \in \Sigma$, it recurses as $\mathsf{string}_\Sigma \vdash P' : \mathsf{string}_\Gamma$
    - If it receives \$, it continues as $\mathbf{1} \vdash P' : \mathsf{string}_\Gamma$
  - Send a label to the right
    - If it sends $a \in \Gamma$, it recurses as $\mathsf{string}_\Sigma \vdash P' : \mathsf{string}_\Gamma$
    - If it sends \$, it continus as $\mathsf{string}_\Sigma \vdash P' : \mathbf{1}$
- $\mathbf{1} \vdash P : \mathsf{string}_\Gamma$ can send finalizing output, then terminates
- $\mathsf{string}_\Sigma \vdash P : \mathbf{1}$ can finish reading input, then terminates

Asynchronous Output

- Typed asynchronous output is already representable

Asynchronous	Synchronous
$\mathsf{R}.k \mathbin{;} P$	$P \mid (\mathsf{R}.k \mathbin{;} \leftrightarrow)$
$\ldots \mid_{\Delta} (\mathsf{R}.k \mathbin{;} P) \mid_{\oplus A_\ell} \ldots$	$\ldots \mid_{\Delta} P \mid_{A_k} (\mathsf{R}.k \mathbin{;} \leftrightarrow) \mid_{\oplus A_\ell} \ldots$

- At the cost of one cut and one identity
- Then $\mathsf{R}.k \mathbin{;} \leftrightarrow$ represents a message
  - So $\ulcorner a \urcorner = \mathsf{R}.a \mathbin{;} \leftrightarrow$ is possible
  - Works also for full session types [DeYoung et al. 2012]
- From synchronous to asynchronous by one commuting conversion and a cut/identity reduction

$$P \mid (\mathsf{R}.k \mathbin{;} \leftrightarrow) \longrightarrow \mathsf{R}.k \mathbin{;} (P \mid \leftrightarrow) \longrightarrow \mathsf{R}.k \mathbin{;} P$$

Composition of Transducers

Theorem (Cut Elimination [Fortier & Santocanale 2013])

If $\Delta \vdash P : A$ and P is a fixed-cut circular proof then there is a cut-free circular proof Q with $\Delta \vdash Q : A$.

Theorem (Closure of SFTs under Composition)

If T and T' are two SFTs with appropriately matching alphabets, there there is an SFT $T \mathbin{;} T'$ which applies T' to the output of T.

Proof.

Let P and P' be the corresponding fixed-cut proofs with $\mathsf{string}_\Sigma \vdash P : \mathsf{string}_\Gamma$ and $\mathsf{string}_\Gamma \vdash P' : \mathsf{string}_\Theta$. Then $\mathsf{string}_\Sigma \vdash (P \mid P') : \mathsf{string}_\Theta$ and, by cut elimination, there is cut-free proof Q with $\mathsf{string}_\Sigma \vdash Q : \mathsf{string}_\Theta$. Construct $T \mathbin{;} T'$ from Q. $\square$

Encoding DFAs

- For composition of SFTs, we can run their programs concurrently, passing messages between them from left to right
- We can establish a bijection between DFAs and processes

$$\mathsf{string}_\Sigma \vdash P : \oplus\{\mathsf{acc} : \mathbf{1}, \mathsf{rej} : \mathbf{1}\}$$

- By allowing multiple endmarkers instead of just \$, one theorem suffices (see paper)

Regular Languages as Types

- For type-checking, can assume for inputs and guarantee for outputs that they adhere to regular language specifications
- Example: no runs of b's

$$\begin{array}{rcl} s_0 & = & \oplus\{a : s_0, b : s_1, \$: \mathbf{1}\} \\ s_1 & = & \oplus\{a : s_0, \qquad\quad \$: \mathbf{1}\} \end{array}$$

- Example: standard bit strings, without leading 0's

$$\begin{array}{rcl} \mathsf{std} & = & \oplus\{0 : \mathsf{pos}, 1 : \mathsf{std}, \$: \mathbf{1}\} \\ \mathsf{pos} & = & \oplus\{0 : \mathsf{pos}, 1 : \mathsf{std} \qquad\quad\} \end{array}$$

Completing Subsingleton Logic

- Adding rules for $A \mathbin{\&} B$

$$\frac{\Delta \vdash A \quad \Delta \vdash B}{\Delta \vdash A \mathbin{\&} B}\ \&R \qquad \frac{A \vdash C}{A \mathbin{\&} B \vdash C}\ \&L_1 \quad \frac{B \vdash C}{A \mathbin{\&} B \vdash C}\ \&L_2$$

- Now $\&L_i$ send, $\&R$ receives
- Labeled versions

$$\frac{\Delta \vdash A_\ell \quad (\forall \ell \in L)}{\Delta \vdash \&_{\ell \in L}\{\ell : A_\ell\}}\ \&R \qquad \frac{A_k \vdash C \quad (k \in L)}{\&_{\ell \in L}\{\ell : A_\ell\} \vdash C}\ \&L_k$$

Completing the Process Language

- New (symmetric) process expressions

$$\frac{\Delta \vdash P_\ell : A_\ell \quad (\forall \ell \in L)}{\Delta \vdash \mathsf{caseR}\,(\ell \Rightarrow P_\ell)_{\ell \in L} : \&_{\ell \in L}\{\ell : A_\ell\}}\ \&R$$

$$\frac{A_k \vdash Q : C \quad (k \in L)}{\&_{\ell \in L}\{\ell : A_\ell\} \vdash \mathsf{L}.k \;;\; Q : C}\ \&L_k$$

- New computation rule

$$\mathsf{caseR}\,(\ell \Rightarrow P_\ell)_{\ell \in L} \mid (\mathsf{L}.k \;;\; Q) \longrightarrow P_k \mid Q$$

- Process expressions now:

$$\begin{array}{rcll}
P, Q & ::= & (P \mid Q) & \text{cut} \\
& \mid & \leftrightarrow & \text{id} \\
& \mid & \mathsf{R}.k \;;\; P \mid \mathsf{caseL}\,(\ell \Rightarrow Q_\ell)_{\ell \in L} & \oplus \\
& \mid & \mathsf{closeR} \mid \mathsf{waitL} \;;\; Q & \mathbf{1} \\
& \mid & \mathsf{caseR}\,(\ell \Rightarrow P_\ell)_{\ell \in L} \mid \mathsf{L}.k \;;\; Q & \&
\end{array}$$

Turing Machines

- First: in mixed string rewriting form
- Transition function $\delta(q, a) = (q', b, \text{left})$ or (q', b, right)
- For each state q, we have two versions
  - q_L, looking left
  - q_R, looking right
- Transition rules

$$
\begin{array}{lcll}
a\, q_L & \longrightarrow & q'_L\, b & \text{if } \delta(q, a) = (q', b, \text{left}) \\
a\, q_L & \longrightarrow & b\, q'_R & \text{if } \delta(q, a) = (q', b, \text{right}) \\
\$\, q_L & \longrightarrow & \$ \sqcup q_L & \text{for endmarker } \$, \text{ blank symbol } \sqcup \\[6pt]
q_R\, a & \longrightarrow & q'_L\, b & \text{if } \delta(q, a) = (q', b, \text{left}) \\
q_R\, a & \longrightarrow & b\, q'_R & \text{if } \delta(q, a) = (q', b, \text{right}) \\
q_R\, \$ & \longrightarrow & q_R \sqcup \$ & \text{for endmarker } \$, \text{ blank symbol } \sqcup
\end{array}
$$

Turing Machines in Subsingleton Logic

- Typing: we must be able to read symbols to the left and right of the read/write head

$$\begin{array}{rcl} \mathsf{tape}_\Sigma & = & \oplus_{a\in\Sigma}\{a : \mathsf{tape}, \$: \mathbf{1}\} \\ \mathsf{epat}_\Sigma & = & \&_{a\in\Sigma}\{a : \mathsf{epat}, \$: \bot\} \end{array}$$

- Program encodes transition

$$\begin{array}{lll} q_L = \mathsf{caseL}\ (\, a \Rightarrow q'_L \mid (\mathsf{L}.b \,;\leftrightarrow) & & \text{if } \delta(q,a) = (q', b, \text{left}) \\ \qquad\qquad\quad \mid a' \Rightarrow (\mathsf{R}.b' \,;\leftrightarrow) \mid q'_R & & \text{if } \delta(q,a') = (q', b', \text{right}) \\ \qquad\qquad\quad \mid \$ \Rightarrow (\mathsf{R}.\$ \,;\leftrightarrow) \mid (\mathsf{R}.\llcorner\!\lrcorner \,;\leftrightarrow) \mid q_L) & & \end{array}$$

$$\begin{array}{lll} q_R = \mathsf{caseR}(\, a \Rightarrow q'_L \mid (\mathsf{L}.b \,;\leftrightarrow) & & \text{if } \delta(q,a) = (q', b, \text{left}) \\ \qquad\qquad\quad \mid a' \Rightarrow (\mathsf{R}.b' \,;\leftrightarrow) \mid q'_R & & \text{if } \delta(q,a') = (q', b', \text{right}) \\ \qquad\qquad\quad \mid \$ \Rightarrow (q_R \mid (\mathsf{L}.\llcorner\!\lrcorner \,;\leftrightarrow) \mid (\mathsf{L}.\$ \,;\leftrightarrow)) & & \end{array}$$

- For halting state, see paper

Turing Machines

- Proofs require embedded cuts, identity
- No longer satisfy circularity condition
  - Proofs are recursive, not coinductive
- Still, steps are simulated faithfully
- No isomorphism: many processes of the right type do not correspond to Turing machines
- Generalize Turing machine model!

A Concurrent Model: LCA

- Linear Communicating Automata
- Similar to Turing machines
  - Multiple read/write heads
  - Can spawn or terminate heads
- Mixed string rewriting of configuration with arbitrary interleaving of alphabet symbols and state symbols
- Distinguish 6 sets of states $q^{\{r,w\}}_{\{L,R\}}$, q_S, q_H

$$
\begin{array}{llllll}
a\,q & \longrightarrow & q' & \text{read left} & \mathsf{caseL}\,(\ldots \mid a \Rightarrow Q' \mid \ldots) \\
q\,a & \longrightarrow & q' & \text{read right} & \mathsf{caseR}\,(\ldots \mid a \Rightarrow Q' \mid \ldots) \\
q & \longrightarrow & a\,q' & \text{write left} & \mathsf{L}.a \,;\, Q' \\
q & \longrightarrow & q'\,a & \text{write right} & \mathsf{R}.a \,;\, Q' \\
q & \longrightarrow & q_1\,q_2 & \text{spawn} & (Q_1 \mid Q_2) \\
q & \longrightarrow & \cdot & \text{halt} & \leftrightarrow \text{ or closeR or closeL}
\end{array}
$$

Well-Typed LCAs Don't Go Wrong

- LCAs can exhibit deadlock and race conditions

Potential deadlock	$q_L^r \; a \; q_R^r$
Potential race	$q_R^r \; a \; q_L^r$

- Use asynchronous representation of configuration

$$\begin{array}{rcl} \ulcorner a \urcorner & = & \mathsf{L}.a \,;\, \leftrightarrow \quad \text{or} \\ \ulcorner a \urcorner & = & \mathsf{R}.a \,;\, \leftrightarrow \end{array}$$

- Type LCAs like we would type their process expressions
- Concurrent, but no race conditions or deadlock

Summary, in Reverse

- Linear communicating automata (LCAs) as concurrent Turing machines

Summary, in Reverse

- Linear communicating automata (LCAs) as concurrent Turing machines
- Subsingleton logic types LCAs, just as intuitionistic logic types λ-calculus

Summary, in Reverse

- Linear communicating automata (LCAs) as concurrent Turing machines
- Subsingleton logic types LCAs, just as intuitionistic logic types λ-calculus
  - Types as regular languages and direction

Summary, in Reverse

- Linear communicating automata (LCAs) as concurrent Turing machines
- Subsingleton logic types LCAs, just as intuitionistic logic types λ-calculus
  - Types as regular languages and direction
  - Proofs as concurrent automata

Summary, in Reverse

- Linear communicating automata (LCAs) as concurrent Turing machines
- Subsingleton logic types LCAs, just as intuitionistic logic types λ-calculus
  - Types as regular languages and direction
  - Proofs as concurrent automata
  - Proof reduction as communication

Summary, in Reverse

- Linear communicating automata (LCAs) as concurrent Turing machines
- Subsingleton logic types LCAs, just as intuitionistic logic types λ-calculus
  - Types as regular languages and direction
  - Proofs as concurrent automata
  - Proof reduction as communication
- Isomorphism for subseq. finite-state transducers (SFTs)

Summary, in Reverse

- Linear communicating automata (LCAs) as concurrent Turing machines
- Subsingleton logic types LCAs, just as intuitionistic logic types λ-calculus
  - Types as regular languages and direction
  - Proofs as concurrent automata
  - Proof reduction as communication
- Isomorphism for subseq. finite-state transducers (SFTs)
  - Use fixed-cut proofs only
  - Encompasses deterministic finite state automata (DFAs)
  - Closure properties via cut elimination

Further Related Work

- Multiparty session types and communicating automata [Deniélou & Yoshida 2012]
- Undecidability of asynchronous session subtyping [Lange & Yoshida] [Bravetti, Carbone, & Zavattaro]
- Many other papers on session types [Honda 1993] [. . .]
- Logical foundations of session types [Caires & Pf 2010]

Future Work

- Develop and apply subsingleton type theory to reason about automata
- Deterministic pushdown automata (DPDAs) and type constructors [DeYoung 2016]
- Parallel cost semantics [Silva & Pf 2016] and analysis
- Other constructions on automata via cut elimination
- Nondeterministic automata via redundant proofs
- Inductive/coinductive/recursive types
- Inductive/coinductive/recursive proofs

Church and Turing

Computation	**Logic**	**Synthesis**
λ-Calculus [Church 1936]	Intuitionistic Logic [Heyting 1930]	Proofs as Programs [Howard 1969]
Turing Machines [Turing 1937]	?	?
Linear Communicating Automata	Subsingleton Logic [Santocanale 2001]	Substructural Proofs as Automata [this paper]
Subsequential Finite State Transducers	Fixed Cut Subsingleton Logic	[this paper]

Church and Turing

Computation	**Logic**	**Synthesis**
λ-Calculus [Church 1936]	Intuitionistic Logic [Heyting 1930]	Proofs as Programs [Howard 1969]
Turing Machines [Turing 1937]	Some Restriction of Subsingleton Logic?	?
Linear Communicating Automata	Subsingleton Logic [Santocanale 2001]	Substructural Proofs as Automata [this paper]
Subsequential Finite State Transducers	Fixed Cut Subsingleton Logic	[this paper]