On the Computational Complexity of Cut-Elimination in Linear Logic

(Joint work with Harry Mairson)

Kazushige Terui

terui@nii.ac.jp

National Institute of Informatics, Tokyo

Motivation

- Cut-elimination in Intuitionistic Logic corresponds to functional computation via Curry-Howard isomorphism.
- Linear Logic decomposes Intuitionistic Logic into Multiplicatives, Additives and Exponentials.
- Thus Linear Logic should decompose functional computation into three.
- But how?
- Address this question from the viewpoint of computational complexity.

Summary

- MLL: PTIME-complete
  — Fulfils all finite computations as efficient as boolean circuits.
- MALL: coNP-complete
  — Nondeterministic cut-elimination with slices.
- Generic MLL: captures PTIME (in terms of realizability)
  — A logical notion of uniformity.
- By-product: Soft Linear Logic without additives captures PTIME
  — Affirmative answer to Lafont's conjecture (Lafont 2001).
- (So does Light Linear Logic.)

Cut-Elimination as a Problem

- Cut-Elimination Problem (CEP):
  Given 2 proofs, do they reduce to the same normal form?
- Subsumes:
  Given a proof π, does it reduce to “true”?
- CEP for Linear Logic is non-elementary (Statman 1979).
- CEP for **MLL** is in PTIME.

Syntax of MLL

- For simplicity, we only consider the intuitionistic fragment.
- Identify **IMLL** proofs = untyped linear lambda terms.
- Justified by Hindley's theorem: any linear lambda term has a simple (propositional) type.
- Types ($\multimap, \forall$) are used neither for restriction nor for enrichment, but for classification.

$$\frac{}{x:A \vdash x:A} \qquad \frac{\Gamma \vdash u:A \quad x:A,\Delta \vdash t:C}{\Gamma,\Delta \vdash t[u/x]:C}$$

$$\frac{x:A,\Gamma \vdash t:B}{\Gamma \vdash \lambda x.t:A \multimap B} \qquad \frac{\Gamma \vdash u:A \quad x:B,\Delta \vdash t:C}{\Gamma, y:A \multimap B,\Delta \vdash t[yu/x]:C}$$

$$\frac{\Gamma \vdash t:A}{\Gamma \vdash t:\forall\alpha.A}\ \alpha \notin FV(\Gamma) \qquad \frac{x:A[B/\alpha],\Gamma \vdash t:C}{x:\forall\alpha.A,\Gamma \vdash t:C}$$

Defined Connectives

$$\mathbf{1} \equiv \forall\alpha.\alpha \multimap \alpha \qquad A \otimes B \equiv \forall\alpha.(A \multimap B \multimap \alpha) \multimap \alpha$$

$$\mathsf{I} \equiv \lambda x.x \qquad t \otimes u \equiv \lambda x.xtu$$

$$\text{let } t \text{ be } \mathsf{I} \text{ in } u \equiv tu \qquad \text{let } t \text{ be } x \otimes y \text{ in } u \equiv t(\lambda xy.u).$$

The above definitions are sound w.r.t.

$$\text{let } \mathsf{I} \text{ be } \mathsf{I} \text{ in } t \longrightarrow t \qquad \text{let } t \otimes u \text{ be } x \otimes y \text{ in } v \longrightarrow v[t/x, u/y]$$

(but *not* w.r.t. the commuting reduction rules)

$$\frac{}{\vdash \mathsf{I} : \mathbf{1}} \qquad \frac{\Gamma \vdash t : C}{x : \mathbf{1}, \Gamma \vdash \text{let } x \text{ be } \mathsf{I} \text{ in } t : C}$$

$$\frac{\Gamma \vdash t : A \quad \Delta \vdash u : B}{\Gamma, \Delta \vdash t \otimes u : A \otimes B} \qquad \frac{x : A, y : B, \Gamma \vdash t : C}{z : A \otimes B, \Gamma \vdash \text{let } z \text{ be } x \otimes y \text{ in } t : C}$$

Π_1 and $e\Pi_1$ types

- Π_1: constructed by $\multimap$, $\otimes$, **1** (viewed as primitives) and positive $\forall$.
- Example: **B** $\equiv \forall\alpha.\alpha \multimap \alpha \multimap \alpha \otimes \alpha$ (multiplicative boolean type)
- Π_1 includes finite data types.
- $e\Pi_1$: like Π_1, but may contain negative inhabited types.
- Example: **B** is Π_1 inhabited. Hence **B** $\multimap$ **B** is $e\Pi_1$.
- $e\Pi_1$ includes functionals over finite data types.

Elimination of $\otimes$ and $\mathbf{1}$

- Proposition: Any Π_1 type is "isomorphic" to another Π_1 type not containing $\otimes$ nor $\mathbf{1}$. Similarly for $e\Pi_1$.
- Proof: Positive $\otimes$ and $\mathbf{1}$ are removed by their Π_1 definitions, while negative ones are removed by

$$((A \otimes B) \multimap C) \quad \circ\!\!-\!\!\circ \quad (A \multimap B \multimap C)$$

$$(\mathbf{1} \multimap C) \quad \circ\!\!-\!\!\circ \quad C$$

Weakening in MLL

- Theorem ($e\Pi_1$-Weakening): For any closed $e\Pi_1$ type A, there is a term w_A of type $A \multimap \mathbf{1}$.
- Examples:

$$
\dfrac{\vdash \mathbf{1} \qquad \dfrac{\vdash \mathbf{1} \qquad \dfrac{\dfrac{\dfrac{\vdash \mathbf{1}}{\mathbf{1} \vdash \mathbf{1}}}{\mathbf{1}, \mathbf{1} \vdash \mathbf{1}}}{\mathbf{1} \otimes \mathbf{1} \vdash \mathbf{1}}}{\mathbf{1} \multimap \mathbf{1} \otimes \mathbf{1} \vdash \mathbf{1}}}{\dfrac{\mathbf{1} \multimap \mathbf{1} \multimap \mathbf{1} \otimes \mathbf{1} \vdash \mathbf{1}}{\mathbf{B} \vdash \mathbf{1}}}
\qquad
\dfrac{\overline{\vdash \mathbf{B}} \qquad \overline{\mathbf{B} \vdash \mathbf{1}}}{\mathbf{B} \multimap \mathbf{B} \vdash \mathbf{1}}
$$

Contraction in MLL

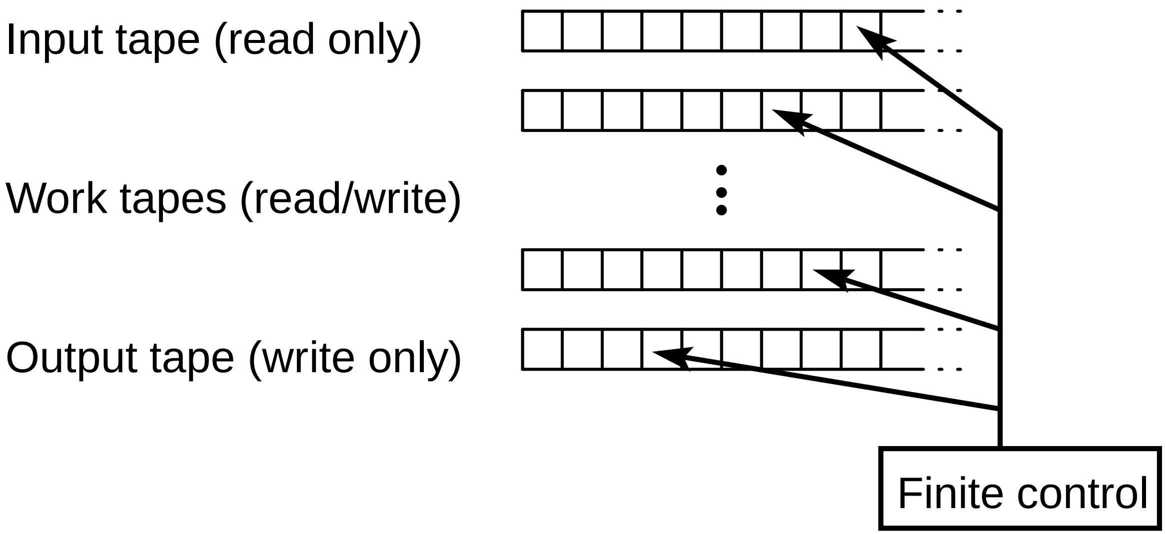
- Theorem(Π_1-Contraction): Let A be a closed inhabited Π_1 type (i.e. data type). Then there is a contraction map $\mathsf{cntr}_A : A \multimap A \otimes A$ such that for any closed term $t : A$,

$$\mathsf{cntr}_A(t) \longrightarrow^* t' \otimes t',$$

where t' is $\beta\eta$-equivalent to t.

Turing Machines and Logspace Functions

- Multi-Tape Turing Machines

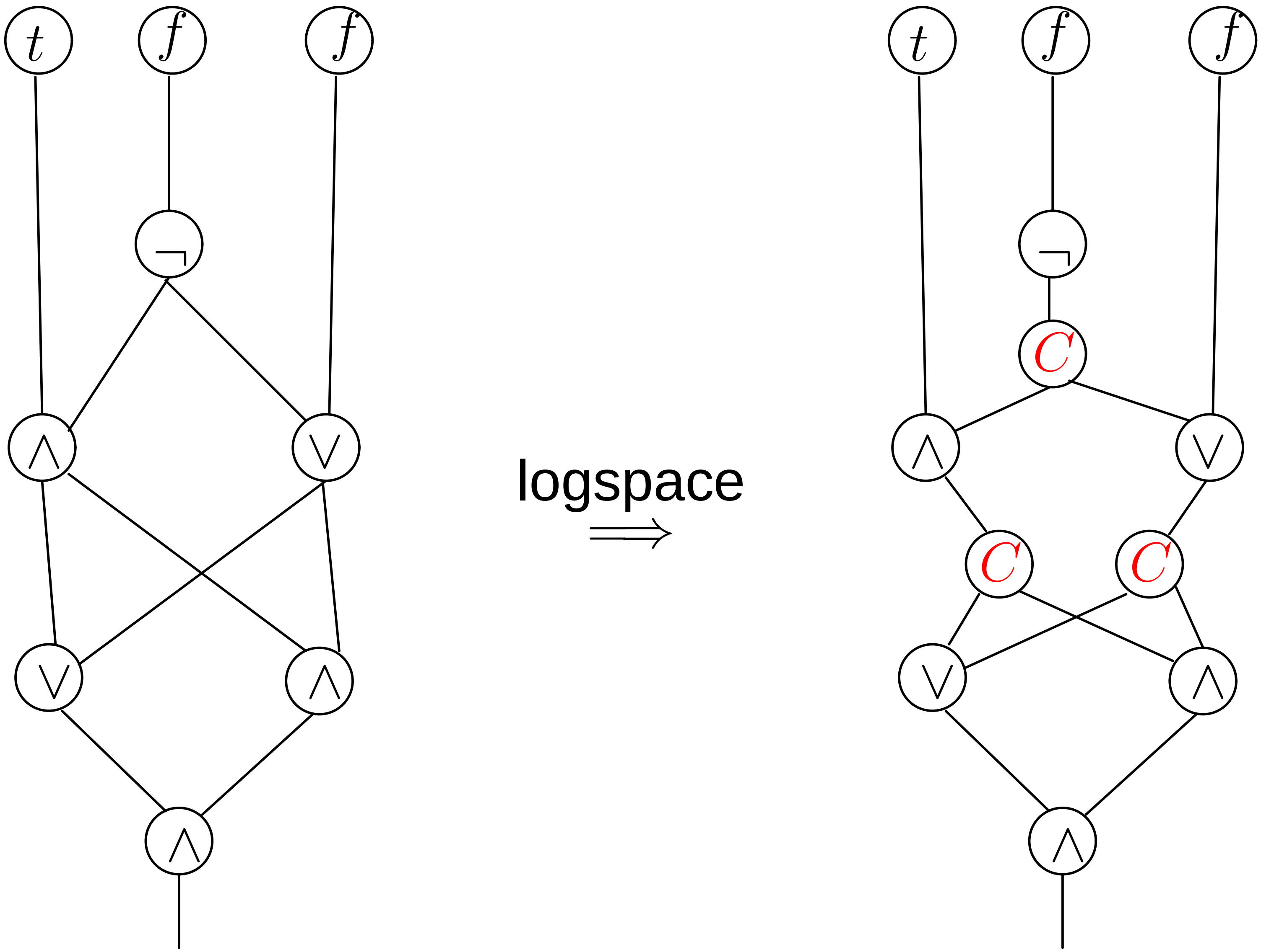


- $f : \{0,1\}^* \longrightarrow \{0,1\}^*$ is logspace if $f(w)$ can be computed within $O(\log n)$ workspace where $n = |w|$.
- Output may be polynomially large.
- The num of all possible config $= O(2^{k \log n}) = O(n^k)$ for some k. In particular, $L \subseteq P$.

PTIME-completeness

- A language $X \subseteq \{0,1\}^*$ is logspace reducible to $Y \subseteq \{0,1\}^*$ if there exists a logspace function $f : \{0,1\}^* \longrightarrow \{0,1\}^*$ such that $w \in X \Leftrightarrow f(w) \in Y$.
- X is PTIME-complete if $X \in$ PTIME and each $Y \in$ PTIME is logspace reducible to X.
- The hardest problems in PTIME.
- If X is PTIME-complete, then $X \notin L$ unless $L =$ PTIME.
- Circuit Value Problem (PTIME-complete, Ladner 1975): Given a boolean circuit C with n inputs and 1 output, and n truth values $\vec{x} = x_1, \ldots, x_n$, is $\vec{x}$ accepted by C?

Boolean Circuits: implicit vs. explicit sharing



Boolean Circuits in MLL

- Projection: for any $e\Pi_1$ type C,

$$\mathsf{fst}_C \equiv \lambda x.\mathsf{let}\ x\ \mathsf{be}\ y \otimes z\ \mathsf{in}\ \left(\mathsf{let}\ \mathsf{w}_C(z)\ \mathsf{be}\ \mathsf{I}\ \mathsf{in}\ y\right)$$

- For any closed term $t \otimes u : A \otimes C$, $\mathsf{fst}_C(t \otimes u) \longrightarrow^* t$.
- Boolean values and connectives:

$$\begin{array}{lcll}
\mathsf{true} & \equiv & \lambda xy.x \otimes y & : \mathbf{B} \\
\mathsf{false} & \equiv & \lambda xy.y \otimes x & : \mathbf{B} \\
\mathsf{not} & \equiv & \lambda Pxy.Pyx & : \mathbf{B} \multimap \mathbf{B} \\
\mathsf{or} & \equiv & \lambda PQ.\mathsf{fst}_{\mathbf{B}}(P\,\mathsf{true}\,Q) & : \mathbf{B} \multimap \mathbf{B} \multimap \mathbf{B} \\
\mathsf{w}_{\mathbf{B}} & \equiv & \lambda z.\mathsf{let}\ z\mathsf{II}\ \mathsf{be}\ x \otimes y\ \mathsf{in}\ \left(\mathsf{let}\ y\ \mathsf{be}\ \mathsf{I}\ \mathsf{in}\ x\right) & : \mathbf{B} \multimap \mathbf{1} \\
\mathsf{cntr} & \equiv & \lambda P.\mathsf{fst}_{\mathbf{B}\otimes\mathbf{B}}(P(\mathsf{true} \otimes \mathsf{true})(\mathsf{false} \otimes \mathsf{false})) & : \mathbf{B} \multimap \mathbf{B} \otimes \mathbf{B}
\end{array}$$

Conditional

- Lemma ($e\Pi_1$-Conditional): Let

$$x_1 : C_1, \ldots, x_n : C_n \vdash t_1, t_2 : D$$

and the type $A \equiv C_1 \multimap \cdots C_n \multimap D$ is $e\Pi_1$. Then there is a conditional

$$b : \mathbf{B}, x_1 : C_1, \ldots, x_n : C_n \vdash \text{if } b \text{ then } t_1 \text{ else } t_2 : D,$$

such that $(\text{if true then } t_1 \text{ else } t_2) \longrightarrow t_1$ and $(\text{if false then } t_1 \text{ else } t_2) \longrightarrow t_2$.

- Proof: Let

$$\text{if } b \text{ then } t \text{ else } u \equiv \text{fst}_{\forall\vec{\alpha}.A}(b(\lambda\vec{x}.t)(\lambda\vec{x}.u))\vec{x}$$

PTIME-completeness of MLL

- Theorem (Mairson2003): There is a logspace algorithm which transforms a boolean circuit C with n inputs and m outputs into a term t_C of type $\mathbf{B}^n \multimap \mathbf{B}^m$, where the size of t_C is $O(|C|)$:

$$C \quad \overset{\text{logspace}}{\Longrightarrow} \quad t_C : \mathbf{B}^n \multimap \mathbf{B}^m$$

  As a consequence, the cut-elimination problem for **IMLL** is PTIME-complete.

- Binary words $\{0,1\}^n$ represented by $\mathbf{B}^n$
- Any $f : \{0,1\}^n \longrightarrow \{0,1\}^m$ represented by a term $t_f : \mathbf{B}^n \multimap \mathbf{B}^m$.
- **MLL** captures all finite functions.

Syntax of IMALL

- Terms of **IMALL**: linear lambda terms plus the following;

  (i) if t and u are terms and $FV(t) = FV(u)$, then so is $\langle t, u \rangle$;

  (ii) if t is a term, then so are $\pi_1(t)$ and $\pi_2(t)$.

- Type assignment rules:

$$\frac{\Gamma \vdash t_1 : A_1 \quad \Gamma \vdash t_2 : A_2}{\Gamma \vdash \langle t_1, t_2 \rangle : A_1 \,\&\, A_2} \qquad \frac{x : A_i, \Gamma \vdash t : C}{y : A_1 \,\&\, A_2, \Gamma \vdash t[\pi_i(y)/x] : C} \; i = 1, 2.$$

- Reduction rule: $\pi_i \langle t_1, t_2 \rangle \longrightarrow t_i$, for $i = 1, 2$.

Normalization in IMALL

- Normalization is exponential as it stands; let

$$t_0 \equiv \lambda x.\langle x, x\rangle$$
$$t_{i+1} \equiv \lambda x.t_i\langle x, x\rangle$$

- The size of $nf(t_i)$ is exponential in i; e.g.

$$\begin{array}{rcl} t_2 & \equiv & \lambda x.(\lambda y.(\lambda z.\langle z, z\rangle)\langle y, y\rangle)\langle x, x\rangle \\ & \longrightarrow & \lambda x.(\lambda y.\langle\langle y, y\rangle, \langle y, y\rangle\rangle)\langle x, x\rangle \\ & \longrightarrow & \lambda x.\langle\langle\langle x, x\rangle, \langle x, x\rangle\rangle, \langle\langle x, x\rangle, \langle x, x\rangle\rangle\rangle \end{array}$$

- How to avoid exponential explosion?
- *Either* restrict to lazy additives (with no positive $\&$ in the conclusion type)
- *Or* adopt nondeterministic cut-elimination with slices

Slices

- A slice of a term t is obtained by slicing:

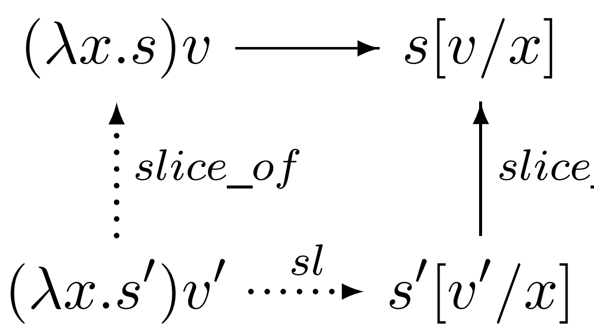
  $\langle u, v\rangle \mapsto \langle u\rangle_1$, or $\langle u, v\rangle \mapsto \langle v\rangle_2$.

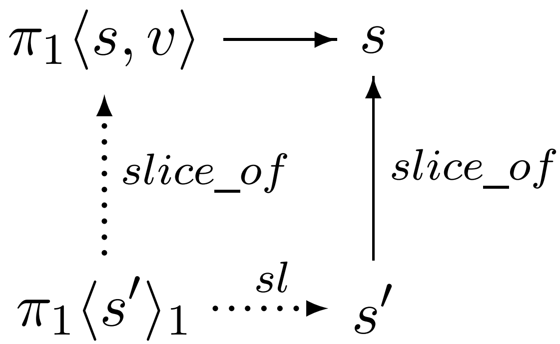
- Two slices t and u (of possibly different terms) are compatible if there is no context (i.e. a term with a hole) Φ such that $t \equiv \Phi[\langle t'\rangle_i]$, $u \equiv \Phi[\langle u'\rangle_j]$, and $i \neq j$.
- Lemma (Slicewise Checking): Two terms t and u are equivalent iff for every compatible pair (t', u') of slices of t and u, we have $t' \equiv u'$.
- Reduction rules for slices:

$$(\lambda x.t)u \xrightarrow{sl} t[u/x], \quad \pi_i\langle t\rangle_i \xrightarrow{sl} t, \quad \pi_i\langle t\rangle_j \xrightarrow{sl} \text{fail}, \quad \text{if } i \neq j.$$

Pullback

- Lemma (Pullback): Let $t \longrightarrow^* u$ and u' be a slice of u. Then there is a unique slice t' of t such that $t' \xrightarrow{sl}{}^* u'$.
- Proof:





- Syntactic counterpart of linearity of linear maps in coherent semantics:

$$\bigcup a_i \sqsubset X \Longrightarrow F(\bigcup a_i) = \bigcup F(a_i)$$

Nondeterministic Cut-Elimination

- There are exponentially many slices for a given term.
- But once a slice has been chosen, the computation afterwards can be done in linear steps, thus in quadratic time.
- We therefore have a nondeterministic polynomial time cut-elimination procedure, viewing the slicing operation as a nondeterministic reduction rule.
- Every slice of a normal form can be reached from the source term in this way (Pullback Lemma).
- The equivalence of two terms can be checked slicewise (Slicewise Checking Lemma).
- Hence the cut-elimination problem for **IMALL** is in coNP.

Encoding a coNP-complete Problem

- Logical Equivalence Problem (coNP-complete): Given two boolean formulas, are they logically equivalent?
- Given a boolean formula C with n variables, $C \quad \mapsto \quad t_C : \mathbf{B}^{(n)} \multimap \mathbf{B}$ in logspace.
- For each $1 \leq k \leq n$, let

$$\mathsf{ta}_k \equiv \lambda f.\lambda x_1 \cdots x_{k-1}.\langle f \text{ true } x_1 \cdots x_{k-1}, f \text{ false } x_1 \cdots x_{k-1}\rangle,$$

which is of type $\forall\alpha.(\mathbf{B}^{(k)} \multimap \alpha) \multimap (\mathbf{B}^{(k-1)} \multimap \alpha \,\&\, \alpha)$, and define

$$\mathsf{ta}(t_C) \equiv \mathsf{ta}_1(\cdots(\mathsf{ta}_n t_C)\cdots) : \underbrace{\mathbf{B} \,\&\, \cdots \,\&\, \mathbf{B}}_{2^n \ times}.$$

- $\mathsf{ta}(t_C)$ can be built from t_C in $O(\log n)$.

Encoding a coNP-complete Problem (2)

- The normal form of $\mathsf{ta}(t_C)$ consists of 2^n boolean values, each of which corresponds to a 'truth assignment' to the formula C.
- Example: $\mathsf{ta}(\mathsf{or})$

$$\begin{aligned}
\mathsf{ta}_1(\mathsf{ta}_2\mathsf{or}) &\longrightarrow \mathsf{ta}_1(\lambda y_1.\langle \mathsf{or\ true}\ y_1, \mathsf{or\ false}\ y_1\rangle) \\
&\longrightarrow^* \langle\langle \mathsf{or\ true\ true}, \mathsf{or\ true\ false}\rangle, \langle \mathsf{or\ false\ true}, \mathsf{or\ false\ false}\rangle\rangle \\
&\longrightarrow^* \langle\langle \mathsf{true}, \mathsf{true}\rangle, \langle \mathsf{true}, \mathsf{false}\rangle\rangle.
\end{aligned}$$

- Two formulas C and D with n variables are logically equivalent if and only if $\mathsf{ta}(t_C)$ and $\mathsf{ta}(t_D)$ reduce to the same normal form.
- Theorem (coNP-completeness of **IMALL**): The cut-elimination problem for **IMALL** is coNP-complete.

Remark (1)

- We do not claim that *the* complexity of **MALL** is coNP (If we considered the complement of CEP, the result would be NP-completeness).
- We do claim that additives have something to do with nondeterminism.

Remark (2)

- In reality, functional computation is never nondeterministic.
- Nondeterministic computation can be simulated by deterministic one with an exponential overhead:
  $\Longrightarrow$ the complexity theoretic meaning of exponential isomorphism

$$!(A \,\&\, B) \multimap\!\circ\, !A \otimes !B.$$

Towards Infinite

- An **MLL** proof represents a finite function. How can one represent the infinite?
- Analogy: A circuit C represents a finite predicate on $\{0,1\}^n$.
- A family $\{C_n\}_{n\in N}$ of boolean circuits (C_n has n inputs) represents an infinite predicate on $\{0,1\}^*$.
- Given an input w of length n, pick up C_n and evaluate $C_n(w)$.
- Such a family may represent a nonrecursive predicate.
- A family $\{C_n\}_{n\in N}$ is logspace uniform if

$$n \quad \stackrel{O(\log n)\text{ space}}{\Longrightarrow} \quad C_n.$$

- Theorem: $X \in$ PTIME $\Longleftrightarrow$ there is a logspace uniform family $\{C_n\}_{n\in N}$ representing X.

Towards logical uniformity

- We could consider logspace uniform families of **MLL** proofs to capture PTIME.
- But logspace uniformity is not a logical concept!
- Is there a purely logical notion of uniformity?
  $\Longrightarrow$ Generic exponentials (Lafont 2001)

Generic MLL(1)

- Types of **MGLL**:

$$A, B ::= \alpha \mid A \multimap B \mid \forall\alpha.A \mid !A$$

- Type assignment rules: **MLL** with generic promotion

$$\frac{x_1 : A_1, \ldots, x_n : A_n \vdash t : B}{x_1 :!A_1, \ldots, x_n :!A_n \vdash t :!B}$$

- Notation: $A^n \equiv \underbrace{A \otimes \cdots \otimes A}_{n\ times}$, $A^0 \equiv$ **1**.

Interpretation: MGLL $\longrightarrow$ MLL

- For each $n \in N$, define a "functor" $\langle n \rangle$ by

$$\begin{array}{cccc} \langle n \rangle : & \textbf{MGLL} & \longrightarrow & \textbf{MLL} \\ & !A & \mapsto & A^n \\ & \dfrac{\Gamma \vdash A}{!\Gamma \vdash !A} & \mapsto & \dfrac{\Gamma \vdash A}{\Gamma^n \vdash A^n} \end{array}$$

- Proposition (Lafont):
  $t : A$ in **MGLL** $\Longrightarrow t\langle n \rangle : A\langle n \rangle$ in **MLL** for each n.
  $\langle n \rangle$ is compatible with cut-elimination.
- Remark: A more general interpretation from **SLL** to **SLL** itself is given in (Lafont 2001).

Genericity $\Rightarrow$ Logspace uniformity

- Theorem: Let an **MGLL** proof $t : A$ be given. Then

$$n \quad \overset{O(\log n) \text{ space}}{\Longrightarrow} \quad t\langle n\rangle.$$

- Every **MGLL** proof t gives a logspace uniform family of **MLL** proofs.

$$t\langle 1\rangle, t\langle 2\rangle, t\langle 3\rangle, \ldots$$

Representing words in MGLL

- **W** $\equiv \forall\alpha.!(\mathbf{B} \multimap \alpha \multimap \alpha) \multimap \alpha \multimap \alpha$.
- **W** has no proof in **MGLL**.
- **W**$\langle n\rangle \equiv \forall\alpha.(\mathbf{B} \multimap \alpha \multimap \alpha)^n \multimap \alpha \multimap \alpha$ for each $n \in N$.
- **W**$\langle n\rangle$ has proofs $\underline{w}$ representing $w \in \{0,1\}^n$.
- $\underline{010} \equiv \lambda f_1 \otimes f_2 \otimes f_3.\lambda x.(f_1\text{false})(f_2\text{true})(f_3\text{false})x))$: **W**$\langle 3\rangle$

Representing predicates in MGLL(1)

- An **MGLL** proof $t : \mathbf{W}^l \multimap \mathbf{B}$ represents a predicate $X \subseteq \{0,1\}^*$ $\Longleftrightarrow$ for each word w of length n,

$$w \in X \iff t\langle n\rangle(\underbrace{\underline{w} \cdots \underline{w}}_{l\ times}) \to^* \text{true}$$

- Theorem: Every proof $t : \mathbf{W}^l \multimap \mathbf{B}$ in **MGLL** represents a PTIME predicate.
- Proof: Given input w of length n, build $t\langle n\rangle$ in logspace, thus in polynomial time, and normalize $t\langle n\rangle(\underline{w} \cdots \underline{w})$ in quadratic time.

Representing predicates in MGLL(2)

- What about the converse?
- Theorem (Lafont): Every PTIME predicate is representable in **MGLL** with additives.
- Theorem (Mairson-Terui): Every PTIME predicate is representable in **MGLL**.
- Every PTIME predicate can be programmed by a linear λ-term.
- No need to think of sharing in program execution.
  1. Given input of length n, compile t into **MLL** proofnet $t\langle n\rangle$.
  2. Normalize it (no sharing, no duplication, efficiently parallelizable)

Simulation of PTIME Turing Machines (1)

- Polynomial clock n^k : $\mathbf{N} \multimap \mathbf{N}\langle X^k \rangle$
  — Already multiplicative in (Lafont 2001)
- One-step transition : $Conf \multimap Conf$
  — (Lafont 2001) uses additives
- Iteration: $A \multimap !(A \multimap A) \multimap \mathbf{N} \multimap A$
- Initialization, Acceptance-checking
  — OK.
- It suffices to give a multiplicative encoding of one-step transition.

Simulation of PTIME Turing Machines (2)

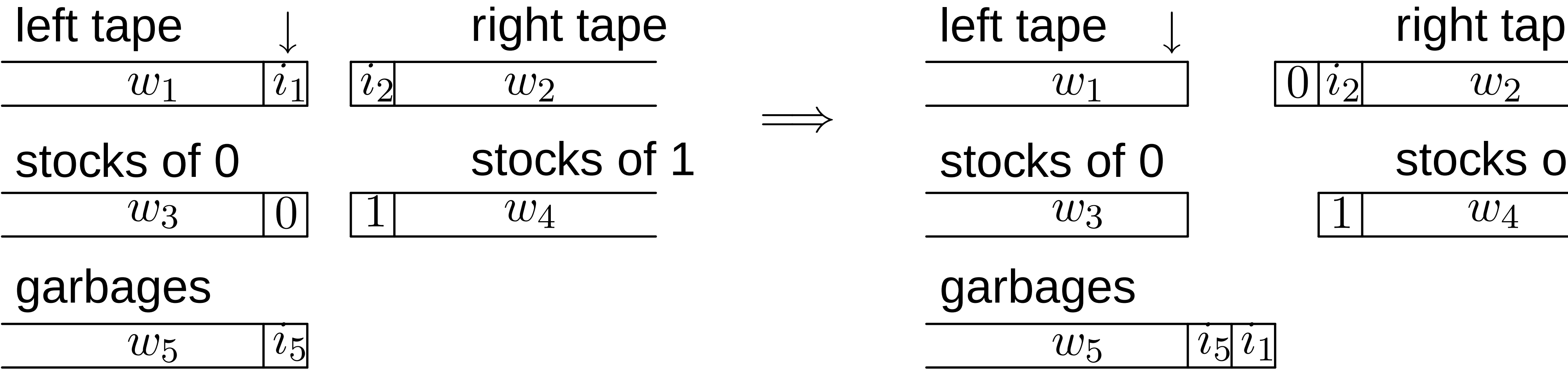
- Consider a TM with 2 symbols and 2^n states. Then,

$$Conf \equiv \forall\alpha.!(\mathbf{B} \multimap \alpha \multimap \alpha) \multimap ((\alpha \multimap \alpha)^5 \otimes \mathbf{B}^n)$$

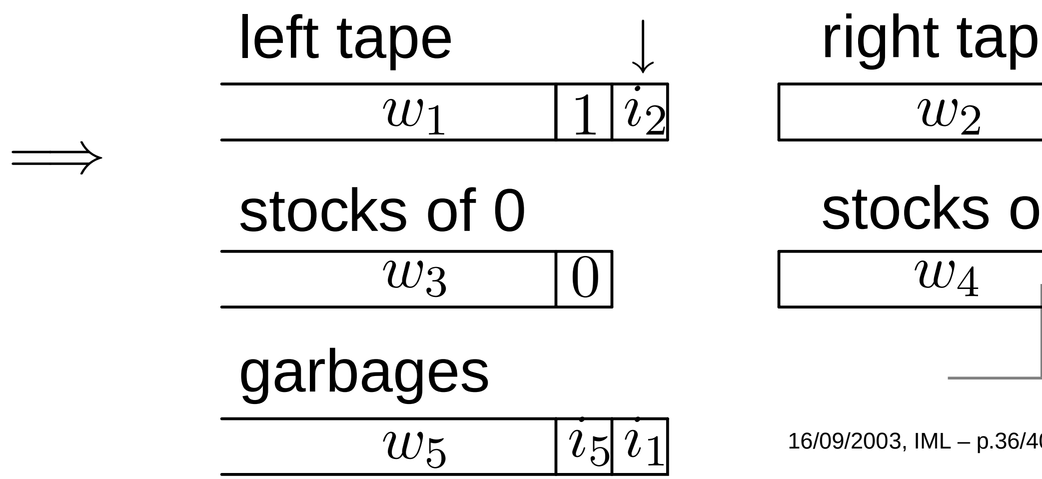
- $\mathbf{B}^n$ corresponds to the 2^n states.
- Usually one needs just 2 stacks $(\alpha \multimap \alpha)^2$, left-tape and right-tape, but we need 5.
- Difficulties:
  1. We cannot create a new tape cell.
  2. We cannot remove a redundant tape cell (as Weakening is available only for closed types).
- Solution: Use 5 stacks to represent each configuration: left-tape, right-tape, stocks of 0, stocks of 1, garbages.

Simulation of PTIME Turing Machines (3)

(1) “Write 0 and move left”



(2) “Write 1 and move right”



Simulation of PTIME Turing machines (4)

- One-step transition is obtained from:
  1. Lafont's ψ function to decompose a stack into the head and the tail
  2. Multiplicative conditional to branch according to the current state
  3. Combinatorial operations to rearrange 5 stacks

Multiplicative Soft Linear Logic

- **MSLL**: **MGLL** with multiplexing (generalization of dereliction, weakening and contraction)

$$!X \multimap X^n, \text{ for each } n \in N$$

- Internalization of $\langle n \rangle$: **MGLL** $\longrightarrow$ **MSLL**
- **W** itself has inhabitants.
- Satisfies polynomial time strong normalization:
  Any proof t of depth d strongly normalizes in time $O(|t|^{d+2})$
  (depth d counts nesting of ! promotions)
- A self-contained logical system of polynomial time (like LLL).

Conclusion

- Multiplicatives: all finite computations (including booleans, conditionals)
- Additives: nondeterminism
- Generic promotion: uniformity
- Soft Linear Logic captures PTIME without additives
- So does Light Linear Logic

Landscape of Complexity in Linear Logic

	Proof Search	Cut-Elimination
MLL	NP-complete	P-complete
MALL	PSPACE-complete	coNP-complete
MSLL	?	EXPTIME-complete
SLL	undecidable	coNEXPTIME-complete(?)
MLLL	?	2EXPTIME-complete
LL	undecidable	Non-Elementary

- Is proof search strictly harder than cut-elimination? : P=NP problem