On Two Extensions of Abstract Categorial Grammars

Philippe de Groote, Sarah Maarek, Ryo Yoshinaka

1. Introduction

Abstract Categorial Grammar

- Type-theoretic grammar formalism for describing natural languages
- Based on the *implicative fragment of linear logic*
- Resource sensitivity

Abstract Categorial Grammar

- Type-theoretic grammar formalism for describing natural languages
- Based on the *implicative fragment of linear logic*
- Resource sensitivity
- Simple but enough expressive
- Mildly context-sensitive languages are generated by second-order ACGs (de Groote and Pogodalla 2004)

Type-Theoretic Extensions of ACGs

- De Groote and Maarek (2007)
- To enable ACGs to, for instance, treat feature structures

Type-Theoretic Extensions of ACGs

- De Groote and Maarek (2007)
- To enable ACGs to, for instance, treat feature structures
  - ACGs with Cartesian product ($\multimap$ $\&$)
  - ACGs with dependent product ($\multimap$ Π)

Type-Theoretic Extensions of ACGs

- De Groote and Maarek (2007)
- To enable ACGs to, for instance, treat feature structures
  - ACGs with Cartesian product ($\multimap$ $\&$)
  - ACGs with dependent product ($\multimap$ Π)
- Those two extensions are Turing-complete formalisms

2. Abstract Categorial Grammars

Types and Terms

- A: a set of *atomic types*

$$\mathcal{T}_A ::= A \mid (\mathcal{T}_A \multimap \mathcal{T}_A)$$

$\alpha \multimap \beta \multimap \gamma \multimap \delta$ abbreviates $(\alpha \multimap (\beta \multimap (\gamma \multimap \delta)))$.

Types and Terms

- A: a set of *atomic types*

$$\mathcal{T}_A ::= A \mid (\mathcal{T}_A \multimap \mathcal{T}_A)$$

$\alpha \multimap \beta \multimap \gamma \multimap \delta$ abbreviates $(\alpha \multimap (\beta \multimap (\gamma \multimap \delta)))$.

- C: a set of *constants*

$$\Lambda_C ::= C \mid x \mid (\lambda^\circ x.\Lambda_C) \mid (\Lambda_C \Lambda_C)$$

$\lambda^\circ xyz.stu$ abbreviates $(\lambda^\circ x.(\lambda^\circ y.(\lambda^\circ z.((st)u))))$.

Typing System

$\Sigma = \langle A, C, \tau \rangle$: a *higher-order signature*

- A: a set of *atomic types*,
- C: a set of *constants*,
- τ: $C \to \mathcal{T}(A)$.

$$\vdash_\Sigma c : \tau(c) \quad \text{for } c \in C$$

$$x : \alpha \vdash_\Sigma x : \alpha \quad \text{for } \alpha \in \mathcal{T}_A$$

$$\frac{\Gamma,\ x : \alpha \vdash_\Sigma t : \beta}{\Gamma \vdash_\Sigma (\lambda^\circ x.t) : \alpha \multimap \beta} \quad x \notin \mathbf{dom}(\Gamma)$$

$$\frac{\Gamma \vdash_\Sigma t : (\alpha \multimap \beta) \quad \Delta \vdash_\Sigma u : \alpha}{\Gamma, \Delta \vdash_\Sigma (tu) : \beta} \quad \mathbf{dom}(\Gamma) \cap \mathbf{dom}(\Delta) = \varnothing$$

Lexicon

- A *lexicon* $\mathscr{L} = \langle \sigma, \theta \rangle$ from $\Sigma_1 = \langle A_1, C_1, \tau_1 \rangle$ to $\Sigma_2 = \langle A_2, C_2, \tau_2 \rangle$:
  - $\sigma : A_1 \rightarrow \mathcal{T}_{A_2}$ (type substitution)
  - $\theta : C_1 \rightarrow \Lambda_{\Sigma_2}$ (term substitution)
  - $\vdash_{\Sigma_2} \theta(c) : \hat{\sigma}(\tau_1(c))$ is provable for all $c \in C_1$, where $\hat{\sigma}$ is the homomorphic extension of σ.
- We write $\mathscr{L}(\cdot)$ for $\hat{\sigma}(\cdot)$ or $\hat{\theta}(\cdot)$, where $\hat{\theta}$ is the homomorphic extension of θ.
- $x_1 : \alpha_1, \ldots, x_m : \alpha_m \vdash_{\Sigma_1} t : \alpha$
  implies
  $x_1 : \mathscr{L}(\alpha_1), \ldots, x_m : \mathscr{L}(\alpha_m) \vdash_{\Sigma_2} \mathscr{L}(t) : \mathscr{L}(\alpha)$,

Abstract Categorial Grammars

An ACG $\mathscr{G} = \langle \Sigma_1, \Sigma_2, \mathscr{L}, s \rangle$:

- Σ_1: *abstract vocabulary*
- Σ_2: *object vocabulary*
- $\mathscr{L}$: a lexicon from Σ_1 to Σ_2
- $s \in A_1$: the distinguished type

Abstract Language:

$$\mathcal{A}(\mathscr{G}) = \{\, t \in \Lambda_{\Sigma_1} \mid \vdash_{\Sigma_1} t : s \text{ is provable} \,\}$$

Object Language:

$$\mathcal{O}(\mathscr{G}) = \{\, t \in \Lambda_{\Sigma_2} \mid \exists u \in \mathcal{A}(\mathscr{G}) \text{ such that } \mathscr{L}(u) = t \,\}$$

We work modulo β:

$$(\lambda^{\circ} x.t)u \rightarrow_{\beta} t[x := u]$$

Abstract Categorial Grammars

An ACG $\mathscr{G} = \langle \Sigma_1, \Sigma_2, \mathscr{L}, s \rangle$:

- Σ_1: *abstract vocabulary*
- Σ_2: *object vocabulary*
- $\mathscr{L}$: a lexicon from Σ_1 to Σ_2
- $s \in A_1$: the distinguished type

Abstract Language:

$$\mathcal{A}(\mathscr{G}) = \{\, t \in \Lambda_{\Sigma_1} \mid \vdash_{\Sigma_1} t : s \text{ is provable} \,\}$$

Object Language:

$$\mathcal{O}(\mathscr{G}) = \{\, t \in \Lambda_{\Sigma_2} \mid \exists u \in \mathcal{A}(\mathscr{G}) \text{ such that } \mathscr{L}(u) = t \,\}$$

Lexicon:

$$\vdash_{\Sigma_2} \mathscr{L}(c) : \mathscr{L}(\tau_1(c))$$

Order of an ACG

- Order of types:
  - $order(p) = 1$ for $p \in A$.
  - $order(\alpha \multimap \beta) = \max\{order(\alpha) + 1,\ order(\beta)\}$ for $\alpha \multimap \beta \in \mathcal{T}_A$.

Order of an ACG

- Order of types:
  - $order(p) = 1$ for $p \in A$.
  - $order(\alpha \multimap \beta) = \max\{order(\alpha) + 1,\ order(\beta)\}$ for $\alpha \multimap \beta \in \mathcal{T}_A$.
- Order of a higher-order signature $\Sigma = \langle A, C, \tau \rangle$:
  - $order(\Sigma) = \max\{\ order(\tau(c)) \mid c \in C\ \}$.

Order of an ACG

- Order of types:
  - $order(p) = 1$ for $p \in A$.
  - $order(\alpha \multimap \beta) = \max\{order(\alpha) + 1,\ order(\beta)\}$ for $\alpha \multimap \beta \in \mathcal{T}_A$.
- Order of a higher-order signature $\Sigma = \langle A, C, \tau \rangle$:
  - $order(\Sigma) = \max\{\ order(\tau(c)) \mid c \in C\ \}$.
- Order of an ACG $\mathscr{G} = \langle \Sigma_1, \Sigma_2, \mathscr{L}, s \rangle$:
  - $order(\mathscr{G}) = order(\Sigma_1)$.

Order of an ACG

- Order of types:
  - $order(p) = 1$ for $p \in A$.
  - $order(\alpha \multimap \beta) = \max\{order(\alpha) + 1,\ order(\beta)\}$ for $\alpha \multimap \beta \in \mathcal{T}_A$.
- Order of a higher-order signature $\Sigma = \langle A, C, \tau \rangle$:
  - $order(\Sigma) = \max\{\ order(\tau(c)) \mid c \in C\ \}$.
- Order of an ACG $\mathscr{G} = \langle \Sigma_1, \Sigma_2, \mathscr{L}, s \rangle$:
  - $order(\mathscr{G}) = order(\Sigma_1)$.

Second-order ACGs generate PTIME languages. (Salvati 2005)
Decidability of membership of Third-order ACGs is open.

Strings by Lambda Terms

- For an alphabet Ξ, let $\Sigma_\Xi = \langle \{o\}, \Xi, \tau \rangle$ be the higher-order signature with $\tau(a) = o \multimap o$ for all $a \in \Xi$.
- A string $a_1 \dots a_k \in \Xi^*$ is represented by the term

$$/a_1 \dots a_k/ = \lambda^\circ z.a_1(\dots(a_k z)\dots).$$

- The empty string ε is represented as $/\varepsilon/ = \lambda^\circ z.z$ in particular.
- Concatenation is realized by $\mathbf{B} = \lambda^\circ xyz.x(yz)$.
- We write $t + u$ for $\lambda^\circ z.t(uz)$.

Example

$c \in C_1$	$\tau_1(c)$	$\mathscr{L}(c)$
S, NP, Adj	$\mathbf{type}$	$o \multimap o$
Pierre	NP	/Pierre/
Etre	$Adj \multimap NP \multimap S$	$\lambda^{\circ} xy.y + /\text{est}/ + x$
Intelligent	Adj	/intelligent/

Example

$c \in C_1$	$\tau_1(c)$	$\mathscr{L}(c)$
S, NP, Adj	$\mathbf{type}$	$o \multimap o$
Pierre	NP	/Pierre/
Etre	$Adj \multimap NP \multimap S$	$\lambda^{\circ} xy.y + /\text{est}/ + x$
Intelligent	Adj	/intelligent/

$$\vdash \textbf{Etre Intelligent Pierre} : S$$

$$\begin{aligned}
&\mathscr{L}(\textbf{Etre})\mathscr{L}(\textbf{Intelligent})\mathscr{L}(\textbf{Pierre}) \\
&= (\lambda^{\circ} xy.y + /\text{est}/ + x)/\text{intelligent}//\text{Pierre}/ \\
&\twoheadrightarrow_{\beta} /\text{Pierre}/ + /\text{est}/ + /\text{intelligent}/
\end{aligned}$$

Example

$c \in C_1$	$\tau_1(c)$	$\mathscr{L}(c)$
S, NP, Adj	$\mathbf{type}$	$o \multimap o$
Pierre	NP	/Pierre/
Etre	$Adj \multimap NP \multimap S$	$\lambda^{\circ} xy.y + /\text{est}/ + x$
Intelligent	Adj	/intelligent/

$$/\text{Pierre}/ + /\text{est}/ + /\text{intelligent}/$$

$$/\text{Marie}/ + /\text{est}/ + /\text{intelligente}/$$

$$/\text{Marie}/ + /\text{et}/ + /\text{Pierre}/ + /\text{sont}/ + /\text{intelligents}/$$

3. ACGs with Dependent Product

ACGs with Dependent Product

- Dependent products are useful in defining generic syntactic categories,

ACGs with Dependent Product

- Dependent products are useful in defining generic syntactic categories,

e.g.,
$$\begin{cases} NP & : \text{noun phrase} \\ NP\,\mathbf{m} & : \text{masculine noun phrase} \\ NP\,\mathbf{f} & : \text{feminine noun phrase} \end{cases}$$

ACGs with Dependent Product

- Dependent products are useful in defining generic syntactic categories,

$$\Sigma_1 \begin{cases} gender : \mathbf{type} \\ S : \mathbf{type} \\ NP : (gender)\mathbf{type} \\ Adj : (gender)\mathbf{type} \\ \hline \mathsf{m} : gender \\ \mathsf{f} : gender \\ \mathsf{Pierre} : NP\,\mathsf{m} \\ \mathsf{Marie} : NP\,\mathsf{f} \\ \mathsf{Etre} : (\Pi x : gender)(Adj\,x \multimap NP\,x \multimap S) \\ \mathsf{Intelligent} : (\Pi x : gender)(Adj\,x) \end{cases}$$

$$\vdash_{\Sigma_1} \mathsf{Etre\,m}: Adj\,\mathsf{m} \multimap NP\,\mathsf{m} \multimap S$$

Kinds, Types, Terms

- Kind:

$$\mathcal{K} ::= \mathbf{type} \mid (\mathcal{T})\mathcal{K}$$

- Type:

$$\mathcal{T} ::= a \mid (\lambda x.\mathcal{T}) \mid (\mathcal{T}\,\Lambda) \mid (\mathcal{T} \multimap \mathcal{T}) \mid (\Pi x : \mathcal{T})\mathcal{T}$$

- Term:

$$\Lambda ::= c \mid x \mid (\lambda^{\circ} x.\Lambda) \mid (\lambda x.\Lambda) \mid (\Lambda\Lambda)$$

Type System

- Type formation rules:

$$\frac{\Gamma \vdash_{\Sigma} \alpha : (\beta_0)(\beta_1)\ldots(\beta_k)\mathbf{type} \quad \Gamma;\vdash_{\Sigma} t : \beta_0}{\Gamma \vdash_{\Sigma} \alpha t : (\beta_1)\ldots(\beta_k)\mathbf{type}}$$

$$\frac{\Gamma, x : \alpha \vdash_{\Sigma} \beta : \mathbf{type}}{\Gamma \vdash_{\Sigma} (\Pi x : \alpha)\beta : \mathbf{type}}$$

Type System

- Type formation rules:

$$\frac{\Gamma \vdash_{\Sigma} \alpha : (\beta_0)(\beta_1)\ldots(\beta_k)\mathbf{type} \quad \Gamma;\vdash_{\Sigma} t : \beta_0}{\Gamma \vdash_{\Sigma} \alpha t : (\beta_1)\ldots(\beta_k)\mathbf{type}}$$

$$\frac{\Gamma, x : \alpha \vdash_{\Sigma} \beta : \mathbf{type}}{\Gamma \vdash_{\Sigma} (\Pi x : \alpha)\beta : \mathbf{type}}$$

Example:

$$\frac{x : gender \vdash_{\Sigma_1} Adj : (gender)\mathbf{type} \quad x : gender;\vdash_{\Sigma_1} x : gender}{\dfrac{x : gender \vdash_{\Sigma_1} Adj\, x : \mathbf{type}}{\vdash_{\Sigma_1} (\Pi x : gender)(Adj\, x) : \mathbf{type}}}$$

Type System

- Typing rules:

$$\frac{\Gamma,\ x : \alpha;\ \Delta \vdash_{\Sigma} t : \beta}{\Gamma; \Delta \vdash_{\Sigma} \lambda x.t : (\Pi x : \alpha)\beta}$$

$$\frac{\Gamma; \Delta \vdash_{\Sigma} t : (\Pi x : \alpha)\beta \quad \Gamma; \vdash_{\Sigma} u : \alpha}{\Gamma; \Delta \vdash_{\Sigma} tu : \beta[x := u]}$$

Type System

- Typing rules:

$$\frac{\Gamma,\ x : \alpha;\ \Delta \vdash_{\Sigma} t : \beta}{\Gamma; \Delta \vdash_{\Sigma} \lambda x.t : (\Pi x : \alpha)\beta}$$

$$\frac{\Gamma; \Delta \vdash_{\Sigma} t : (\Pi x : \alpha)\beta \quad \Gamma; \vdash_{\Sigma} u : \alpha}{\Gamma; \Delta \vdash_{\Sigma} tu : \beta[x := u]}$$

Example:

$$\frac{\vdash_{\Sigma_1} \textsf{Intelligent} : (\Pi x : gender)(Adj\ x) \quad \vdash_{\Sigma_1} \textsf{m} : gender}{\vdash_{\Sigma_1} \textsf{Intelligent m} : Adj\ \textsf{m}}$$

Turing-Completeness

- Since the type system proposed by de Groote and Maarek as the basis of ACGs with dependent product contains the Edinburgh logical framework, we may expect this extension is also Turing complete.

Turing-Completeness

- Since the type system proposed by de Groote and Maarek as the basis of ACGs with dependent product contains the Edinburgh logical framework, we may expect this extension is also Turing complete.
- We show that any recursively enumerable language is generated by an ACG with dependent product

Turing-Completeness

- Since the type system proposed by de Groote and Maarek as the basis of ACGs with dependent product contains the Edinburgh logical framework, we may expect this extension is also Turing complete.
- We show that any recursively enumerable language is generated by an ACG with dependent product
- By reduction from two-stack PDAs, which are known to be equivalent to Turing machines.

Two-Stack Pushdown Automaton

$M = \langle Q, \Gamma, \Xi, \delta, q_0, Z_1, Z_2, q_f \rangle$:

- Q: *states*, Γ: *stack symbols*, Ξ: *input symbols*, $q_f \in Q$: *final state*,

Two-Stack Pushdown Automaton

$M = \langle Q, \Gamma, \Xi, \delta, q_0, Z_1, Z_2, q_f \rangle$:

- Q: *states*, Γ: *stack symbols*, Ξ: *input symbols*, $q_f \in Q$: *final state*,
- A *configuration* is a member of $Q \times \Gamma^* \times \Gamma^*$,

Two-Stack Pushdown Automaton

$M = \langle Q, \Gamma, \Xi, \delta, q_0, Z_1, Z_2, q_f \rangle$:

- Q: *states*, Γ: *stack symbols*, Ξ: *input symbols*, $q_f \in Q$: *final state*,
- A *configuration* is a member of $Q \times \Gamma^* \times \Gamma^*$,
- $\langle q_0, Z_1, Z_2 \rangle \in Q \times \Gamma \times \Gamma$ is the *initial configuration*,

Two-Stack Pushdown Automaton

$M = \langle Q, \Gamma, \Xi, \delta, q_0, Z_1, Z_2, q_f \rangle$:

- Q: *states*, Γ: *stack symbols*, Ξ: *input symbols*, $q_f \in Q$: *final state*,
- A *configuration* is a member of $Q \times \Gamma^* \times \Gamma^*$,
- $\langle q_0, Z_1, Z_2 \rangle \in Q \times \Gamma \times \Gamma$ is the *initial configuration*,
- Each *transition rule* in δ has the form

$$\langle q, X_1, X_2 \rangle \xrightarrow{a} \langle r, \beta_1, \beta_2 \rangle,$$

for $q, r \in Q$, $X_1, X_2 \in \Gamma$, $\beta_1, \beta_2 \in \Gamma^*$ and $a \in \Xi \cup \{\varepsilon\}$.

Two-Stack Pushdown Automaton

$M = \langle Q, \Gamma, \Xi, \delta, q_0, Z_1, Z_2, q_f \rangle$:

- Q: *states*, Γ: *stack symbols*, Ξ: *input symbols*, $q_f \in Q$: *final state*,
- A *configuration* is a member of $Q \times \Gamma^* \times \Gamma^*$,
- $\langle q_0, Z_1, Z_2 \rangle \in Q \times \Gamma \times \Gamma$ is the *initial configuration*,
- Each *transition rule* in δ has the form

$$\langle q, X_1, X_2 \rangle \xrightarrow{a} \langle r, \beta_1, \beta_2 \rangle,$$

  for $q, r \in Q$, $X_1, X_2 \in \Gamma$, $\beta_1, \beta_2 \in \Gamma^*$ and $a \in \Xi \cup \{\varepsilon\}$.

- $\langle q, X_1\gamma_1, X_2\gamma_2 \rangle \xrightarrow[M]{a} \langle r, \beta_1\gamma_1, \beta_2\gamma_2 \rangle$ for any $\gamma_1, \gamma_2 \in \Gamma^*$.

Two-Stack Pushdown Automaton

$M = \langle Q, \Gamma, \Xi, \delta, q_0, Z_1, Z_2, q_f \rangle$:

- Q: *states*, Γ: *stack symbols*, Ξ: *input symbols*, $q_f \in Q$: *final state*,
- A *configuration* is a member of $Q \times \Gamma^* \times \Gamma^*$,
- $\langle q_0, Z_1, Z_2 \rangle \in Q \times \Gamma \times \Gamma$ is the *initial configuration*,
- Each *transition rule* in δ has the form

$$\langle q, X_1, X_2 \rangle \xrightarrow{a} \langle r, \beta_1, \beta_2 \rangle,$$

  for $q, r \in Q$, $X_1, X_2 \in \Gamma$, $\beta_1, \beta_2 \in \Gamma^*$ and $a \in \Xi \cup \{\varepsilon\}$.

- $\langle q, X_1\gamma_1, X_2\gamma_2 \rangle \xrightarrow[M]{a} \langle r, \beta_1\gamma_1, \beta_2\gamma_2 \rangle$ for any $\gamma_1, \gamma_2 \in \Gamma^*$.
- The language accepted by M is given by
  $L(M) = \{ w \mid \langle q_0, Z_1, Z_2 \rangle \xrightarrow[M]{w} \langle q_f, \alpha_1, \alpha_2 \rangle$ for some $\alpha_1, \alpha_2 \in \Gamma^* \}$
  (accept by final state).

Encoding by ACG with dependent product

- $M = \langle Q, \Gamma, \Xi, \delta, q_0, Z_1, Z_2, q_f \rangle$: 2PDA
- $\mathscr{G}_M = \langle \Sigma_M, \Sigma_\Xi, \mathscr{L}, s \rangle$: ACG with dependent product

Encoding by ACG with dependent product

- $M = \langle Q, \Gamma, \Xi, \delta, q_0, Z_1, Z_2, q_f \rangle$: 2PDA
- $\mathscr{G}_M = \langle \Sigma_M, \Sigma_\Xi, \mathscr{L}, s \rangle$: ACG with dependent product

$$\Sigma_M \begin{cases} o : \mathbf{type}, \\ q : (o)(o)\mathbf{type} \quad \text{for all } q \in Q, \\ \hline \# : o, \\ X : o \multimap o \quad \text{for all } X \in \Gamma, \end{cases}$$

Configuration $\langle q, \alpha_1, \alpha_2 \rangle$ of M

$\Updownarrow$

Type $q(/\alpha_1/\#)(/\alpha_2/\#)$ in $\mathscr{G}_M$

Encoding by ACG with dependent product

- $M = \langle Q, \Gamma, \Xi, \delta, q_0, Z_1, Z_2, q_f \rangle$: 2PDA
- $\mathscr{G}_M = \langle \Sigma_M, \Sigma_\Xi, \mathscr{L}, s \rangle$: ACG with dependent product

$$\Sigma_M \begin{cases} o, s : \mathbf{type}, \\ \quad q : (o)(o)\mathbf{type} & \text{for all } q \in Q, \\ \hline \quad \# : o, \\ \quad X : o \multimap o & \text{for all } X \in \Gamma, \\ \quad c_0 : q_0(Z_1\#)(Z_2\#), \\ \quad c_\rho \ : (\Pi x_1, x_2 : o)\big(q(X_1 x_1)(X_2 x_2) \multimap r(/\beta_1/x_1)(/\beta_2/x_2)\big) \\ \qquad\qquad\qquad\qquad \text{for all } \rho = \langle q, X_1, X_2 \rangle \xrightarrow{a} \langle r, \beta_1, \beta_2 \rangle \in \delta, \\ \quad c_f \ : (\Pi x_1, x_2 : o)(q_f x_1 x_2 \multimap s). \end{cases}$$

- $\langle q_0, Z_1, Z_2 \rangle \xrightarrow{w} \langle q, \alpha_1, \alpha_2 \rangle$
  iff $\vdash_{\Sigma_M} t : q(/\alpha_1/\#)(/\alpha_2/\#)$ is provable.

Encoding by ACG with Dependent Product

- $M = \langle Q, \Gamma, \Xi, \delta, q_0, Z_1, Z_2, q_f \rangle$: 2PDA
- $\mathscr{G}_M = \langle \Sigma_M, \Sigma_\Xi, \mathscr{L}, s \rangle$: ACG with dependent product

$$\mathscr{L} \begin{cases} o, s := o \multimap o, \\ \quad q := \lambda x_1 x_2 . o \multimap o \quad \text{for all } q \in Q, \\ \hline \quad \# := /\varepsilon/, \\ \quad X := \lambda^\circ y . y \quad \text{for all } X \in \Gamma, \\ \quad c_0 := /\varepsilon/, \\ \quad c_\rho := \lambda x_1 x_2 . \lambda^\circ y . y + /a/ \\ \qquad\qquad \text{for all } \rho = \langle q, X_1, X_2 \rangle \xrightarrow{a} \langle r, \beta_1, \beta_2 \rangle \in \delta, \\ \quad c_f := \lambda x_1 x_2 . \lambda^\circ y . y. \end{cases}$$

- $\langle q_0, Z_1, Z_2 \rangle \xrightarrow{w} \langle q, \alpha_1, \alpha_2 \rangle$
  iff $\vdash_{\Sigma_M} t : q(/\alpha_1/\#)(/\alpha_2/\#)$ and $\mathscr{L}(t) = /w/$.

$$L(M) = \mathcal{O}(\mathscr{G}_M)$$

Theorem

- For any two-stack pushdown automaton,
  one can construct a second-order ACG with dependent product
  that generates the same language.

Theorem

- For any two-stack pushdown automaton,
  one can construct a second-order ACG with dependent product
  that generates the same language.

$$order((\Pi x : \alpha)\beta) = \max\{order(\alpha) + 1, order(\beta)\}$$

$$order(\alpha \multimap \beta) = \max\{order(\alpha) + 1, order(\beta)\}$$

4. ACGs with Cartesian Product

ACGs with Cartesian Product

- $inflection = \langle gender, number \rangle$,
- $feutures = \langle inflection, function \rangle$,
- $$\left[\begin{array}{lcl} inflection & = & \left[\begin{array}{lcl} gender & = & \textsf{masculine} \\ number & = & \textsf{singular} \end{array}\right] \\ function & = & \textsf{subject} \end{array}\right]$$

ACGs with Cartesian Product

Let $feuture = gender \,\&\, number$.

$$\Sigma_1 \begin{cases} gender, number : & \mathbf{type} \\ S : & \mathbf{type} \\ NP, Adj : & (feuture)\mathbf{type} \\ \hline \mathsf{m}, \mathsf{f} : & gender \\ \mathsf{s}, \mathsf{p} : & number \\ \mathsf{Marie} : & NP\langle \mathsf{f}, \mathsf{s} \rangle \\ \mathsf{LesGens} : & NP\langle \mathsf{m}, \mathsf{p} \rangle \\ \mathsf{Etre} : & (\Pi x : feuture)(Adj\ x \multimap NP\ x \multimap S) \\ \mathsf{Intelligent} : & (\Pi x : feuture)(Adj\ x) \end{cases}$$

ACGs with Cartesian Product

Let $feuture = gender \;\&\; number$.

$$\Sigma_1 \begin{cases} gender, number : & \mathbf{type} \\ S : & \mathbf{type} \\ NP, Adj : & (feuture)\mathbf{type} \\ \hline \mathsf{m}, \mathsf{f} : & gender \\ \mathsf{s}, \mathsf{p} : & number \\ \mathsf{Marie} : & NP\langle \mathsf{f}, \mathsf{s} \rangle \\ \mathsf{LesGens} : & NP\langle \mathsf{m}, \mathsf{p} \rangle \\ \mathsf{Etre} : & (\Pi x : feuture)(Adj\, x \multimap NP\, x \multimap S) \\ \mathsf{Intelligent} : & (\Pi x : feuture)(Adj\, x) \end{cases}$$

$$\begin{cases} /\text{Marie}/ + /\text{est}/ + /\text{intelligente}/ \\ /\text{Les}/ + /\text{gens}/ + /\text{sont}/ + /\text{intelligents}/ \end{cases}$$

ACGs with Cartesian Product

Let $feuture = gender \;\&\; number$.

$$\mathscr{L} \begin{cases} \mathbf{m} := & \lambda^{\circ} y.\pi_1 y \\ \mathbf{f} := & \lambda^{\circ} y.\pi_2 y \\ \mathbf{s} := & \lambda^{\circ} y.\pi_1 y \\ \mathbf{p} := & \lambda^{\circ} y.\pi_2 y \\ \mathbf{Marie} := & /\text{Marie}/ \\ \mathbf{LesGens} := & /\text{Les}/ + /\text{gens}/ \\ \mathbf{Etre} := & \lambda x.\lambda^{\circ} y_1 y_2.y_2 + \pi_2 x\langle /\text{est}/, /\text{sont}/\rangle + y_1 \\ \mathbf{Intelligent} := & \lambda x.\pi_1 x(\pi_2 x\langle\langle /\text{intelligent}/, /\text{intelligente}/\rangle, \\ & \qquad \langle /\text{intelligents}/, /\text{intelligentes}/\rangle\rangle \end{cases}$$

$$\mathscr{L}(\mathbf{Intelligent}\langle \mathbf{f}, \mathbf{p}\rangle) = /\text{intelligentes}/$$

Types, Terms and Reduction Rules

$$\mathcal{T}_A ::= A \mid (\mathcal{T}_A \multimap \mathcal{T}_A) \mid (\mathcal{T}_A \,\&\, \mathcal{T}_A)$$

$$\Lambda_\Sigma ::= C \mid x \mid (\lambda^\circ x.\Lambda_\Sigma) \mid (\Lambda_\Sigma \Lambda_\Sigma) \mid \langle \Lambda_\Sigma, \Lambda_\Sigma \rangle \mid (\pi_1 \Lambda_\Sigma) \mid (\pi_2 \Lambda_\Sigma)$$

$$\pi_1 \langle t, u \rangle \rightarrow_\beta t, \quad \pi_2 \langle t, u \rangle \rightarrow_\beta u$$

Type System

$$\vdash_{\Sigma} c : \tau(c) \quad \text{for } c \in C$$

$$x : \alpha \vdash_{\Sigma} x : \alpha \quad \text{for } \alpha \in \mathcal{T}_A$$

$$\frac{\Gamma,\ x : \alpha \vdash_{\Sigma} t : \beta}{\Gamma \vdash_{\Sigma} (\lambda^{\circ} x.t) : \alpha \multimap \beta} \quad x \notin \mathbf{dom}(\Gamma)$$

$$\frac{\Gamma \vdash_{\Sigma} t : (\alpha \multimap \beta) \quad \Delta \vdash_{\Sigma} u : \alpha}{\Gamma, \Delta \vdash_{\Sigma} (tu) : \beta} \qquad \mathbf{dom}(\Gamma) \cap \mathbf{dom}(\Delta) = \varnothing$$

$$\frac{\Gamma \vdash_{\Sigma} t : \alpha \quad \Gamma \vdash_{\Sigma} u : \beta}{\Gamma \vdash_{\Sigma} \langle t, u \rangle : \alpha \,\&\, \beta}$$

$$\frac{\Gamma \vdash_{\Sigma} t : \alpha \,\&\, \beta}{\Gamma \vdash_{\Sigma} \pi_1 t : \alpha} \qquad \frac{\Gamma \vdash_{\Sigma} t : \alpha \,\&\, \beta}{\Gamma \vdash_{\Sigma} \pi_2 t : \beta}$$

Membership Problem

- Membership of (linear implicative) ACGs $\leftrightarrow$ MELL
  (decidability is open)
- Membership of ACGs with Cartesian product $\overset{?}{\leftrightarrow}$ MAELL
  (Turing complete)
- Any recursively enumerable language is generated by an ACG with Cartesian product

Turing-Completeness

- Computational equivalence between Turing machines and two-counter machines is known.

Two-counter machine

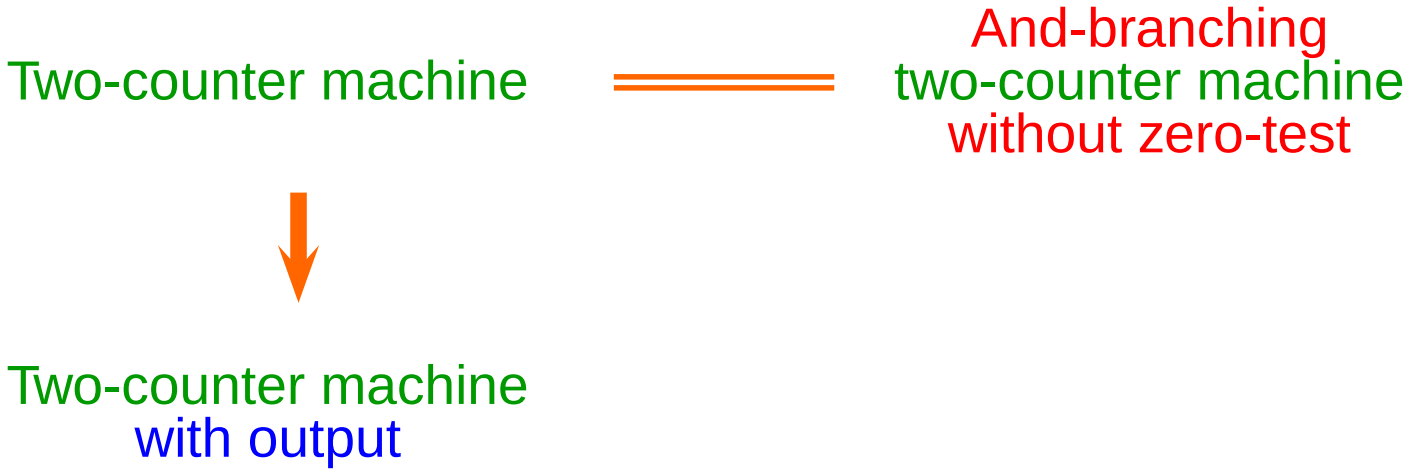
Turing-Completeness

- Computational equivalence between Turing machines and two-counter machines is known.
- Lincoln et al.(1992) show the undecidability of provability of MAELL by reduction from two-counter machines via and-branching two-counter machines without zero-test.

Two-counter machine $\equiv$ And-branching two-counter machine without zero-test

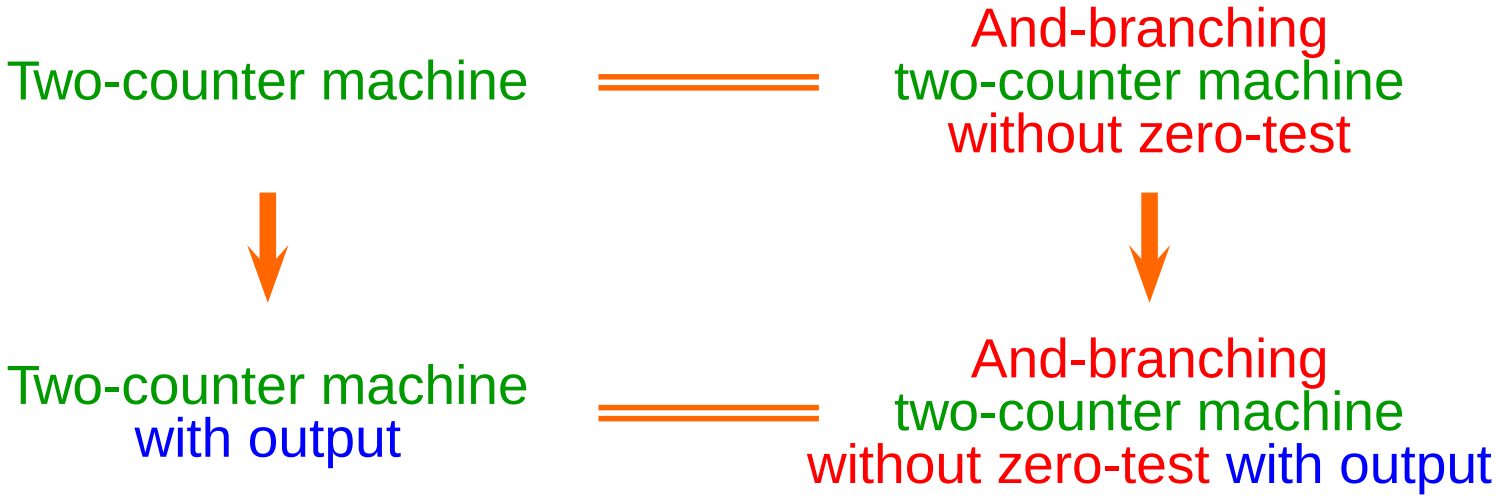
Turing-Completeness

- Computational equivalence between Turing machines and two-counter machines is known.
- Lincoln et al.(1992) show the undecidability of provability of MAELL by reduction from two-counter machines via and-branching two-counter machines without zero-test.
- Two-counter machines compute natural numbers, not strings
- $\longrightarrow$ *two-counter machine with output tape*



Turing-Completeness

- Computational equivalence between Turing machines and two-counter machines is known.
- Lincoln et al.(1992) show the undecidability of provability of MAELL by reduction from two-counter machines via and-branching two-counter machines without zero-test.
- Two-counter machines compute natural numbers, not strings
- $\longrightarrow$ *two-counter machine with output tape*



Two-counter Machine with Output Tape

- $M = \langle Q, \Xi, \delta, q_0, q_f \rangle$:
  - Q: states,
  - Ξ: output letters,
  - $q_0 \in Q$: initial state,
  - $q_f \in Q$: final state,
  - A configuration of M is a member of $Q \times \mathbb{N} \times \mathbb{N}$.

Two-counter Machine with Output Tape

- $M = \langle Q, \Xi, \delta, q_0, q_f \rangle$:
  - Q: states,
  - Ξ: output letters,
  - $q_0 \in Q$: initial state,
  - $q_f \in Q$: final state,
  - A configuration of M is a member of $Q \times \mathbb{N} \times \mathbb{N}$.
  - Each rule in δ has one of the following forms:

$$\langle q\ \mathbf{Inc}\ i\ r \rangle, \quad \langle q\ \mathbf{Dec}\ i\ r \rangle, \quad \langle q\ \mathbf{Zero?}\ i\ r \rangle, \quad \langle q\ \mathbf{Write}\ a\ r \rangle$$

for $q, r \in Q,\ i \in \{1, 2\},\ a \in \Xi$.

Two-counter Machine with Output Tape

- $M = \langle Q, \Xi, \delta, q_0, q_f \rangle$:
  - Q: states,
  - Ξ: output letters,
  - $q_0 \in Q$: initial state,
  - $q_f \in Q$: final state,
  - A configuration of M is a member of $Q \times \mathbb{N} \times \mathbb{N}$.
  - Each rule in δ has one of the following forms:

$$\langle q\ \mathbf{Inc}\ i\ r \rangle, \quad \langle q\ \mathbf{Dec}\ i\ r \rangle, \quad \langle q\ \mathbf{Zero?}\ i\ r \rangle, \quad \langle q\ \mathbf{Write}\ a\ r \rangle$$

for $q, r \in Q,\ i \in \{1, 2\},\ a \in \Xi$.

$$\begin{cases} \langle q, m, n \rangle \to \langle r, m+1, n \rangle & \text{if } \langle q\ \mathbf{Inc}\ 1\ r \rangle \in \delta, \\ \langle q, m, n \rangle \to \langle r, m, n+1 \rangle & \text{if } \langle q\ \mathbf{Inc}\ 2\ r \rangle \in \delta, \\ \langle q, m+1, n \rangle \to \langle r, m, n \rangle & \text{if } \langle q\ \mathbf{Dec}\ 1\ r \rangle \in \delta, \\ \langle q, m, n+1 \rangle \to \langle r, m, n \rangle & \text{if } \langle q\ \mathbf{Dec}\ 2\ r \rangle \in \delta, \end{cases}$$

Two-counter Machine with Output Tape

- $M = \langle Q, \Xi, \delta, q_0, q_f \rangle$:
  - Q: states,
  - Ξ: output letters,
  - $q_0 \in Q$: initial state,
  - $q_f \in Q$: final state,
  - A configuration of M is a member of $Q \times \mathbb{N} \times \mathbb{N}$.
  - Each rule in δ has one of the following forms:

$$\langle q\ \mathbf{Inc}\ i\ r \rangle, \quad \langle q\ \mathbf{Dec}\ i\ r \rangle, \quad \langle q\ \mathbf{Zero?}\ i\ r \rangle, \quad \langle q\ \mathbf{Write}\ a\ r \rangle$$

for $q, r \in Q,\ i \in \{1, 2\},\ a \in \Xi$.

$$\begin{cases} \langle q, m, n \rangle \rightarrow \langle r, m+1, n \rangle & \text{if } \langle q\ \mathbf{Inc}\ 1\ r \rangle \in \delta, \\ \langle q, m+1, n \rangle \rightarrow \langle r, m, n \rangle & \text{if } \langle q\ \mathbf{Dec}\ 1\ r \rangle \in \delta, \end{cases}$$

Two-counter Machine with Output Tape

- $M = \langle Q, \Xi, \delta, q_0, q_f \rangle$:
  - Q: states,
  - Ξ: output letters,
  - $q_0 \in Q$: initial state,
  - $q_f \in Q$: final state,
  - A configuration of M is a member of $Q \times \mathbb{N} \times \mathbb{N}$.
  - Each rule in δ has one of the following forms:

$$\langle q\ \mathbf{Inc}\ i\ r \rangle, \quad \langle q\ \mathbf{Dec}\ i\ r \rangle, \quad \langle q\ \mathbf{Zero?}\ i\ r \rangle, \quad \langle q\ \mathbf{Write}\ a\ r \rangle$$

for $q, r \in Q,\ i \in \{1, 2\},\ a \in \Xi$.

$$\begin{cases} \langle q, m, n \rangle \to \langle r, m+1, n \rangle & \text{if } \langle q\ \mathbf{Inc}\ 1\ r \rangle \in \delta, \\ \langle q, m+1, n \rangle \to \langle r, m, n \rangle & \text{if } \langle q\ \mathbf{Dec}\ 1\ r \rangle \in \delta, \\ \langle q, 0, n \rangle \to \langle r, 0, n \rangle & \text{if } \langle q\ \mathbf{Zero?}\ 1\ r \rangle \in \delta, \end{cases}$$

Two-counter Machine with Output Tape

- $M = \langle Q, \Xi, \delta, q_0, q_f \rangle$:
  - Q: states,
  - Ξ: output letters,
  - $q_0 \in Q$: initial state,
  - $q_f \in Q$: final state,
  - A configuration of M is a member of $Q \times \mathbb{N} \times \mathbb{N}$.
  - Each rule in δ has one of the following forms:

$$\langle q\ \mathbf{Inc}\ i\ r \rangle, \quad \langle q\ \mathbf{Dec}\ i\ r \rangle, \quad \langle q\ \mathbf{Zero?}\ i\ r \rangle, \quad \langle q\ \mathbf{Write}\ a\ r \rangle$$

for $q, r \in Q,\ i \in \{1, 2\},\ a \in \Xi$.

$$\begin{cases} \langle q, m, n \rangle \xrightarrow{\varepsilon} \langle r, m+1, n \rangle & \text{if } \langle q\ \mathbf{Inc}\ 1\ r \rangle \in \delta, \\ \langle q, m+1, n \rangle \xrightarrow{\varepsilon} \langle r, m, n \rangle & \text{if } \langle q\ \mathbf{Dec}\ 1\ r \rangle \in \delta, \\ \langle q, 0, n \rangle \xrightarrow{\varepsilon} \langle r, 0, n \rangle & \text{if } \langle q\ \mathbf{Zero?}\ 1\ r \rangle \in \delta, \\ \langle q, m, n \rangle \xrightarrow{a} \langle r, m, n \rangle & \text{if } \langle q\ \mathbf{Write}\ a\ r \rangle \in \delta. \end{cases}$$

Two-counter Machine with Output Tape

- if $\langle q_1, m_1, n_1 \rangle \xrightarrow{a_1} \langle q_2, m_2, n_2 \rangle \xrightarrow{a_2} \ldots \xrightarrow{a_k} \langle q_{k+1}, m_{k+1}, n_{k+1} \rangle$, we write

$$\langle q_1, m_1, n_1 \rangle \xrightarrow{a_1 \ldots a_k} \langle q_{k+1}, m_{k+1}, n_{k+1} \rangle$$

for $a_1, \ldots, a_k \in \Xi \cup \{\varepsilon\}$, $q_i \in Q$, $m_i, n_i \in \mathbb{N}$.

Two-counter Machine with Output Tape

- if $\langle q_1, m_1, n_1 \rangle \xrightarrow{a_1} \langle q_2, m_2, n_2 \rangle \xrightarrow{a_2} \ldots \xrightarrow{a_k} \langle q_{k+1}, m_{k+1}, n_{k+1} \rangle$, we write

$$\langle q_1, m_1, n_1 \rangle \xrightarrow{a_1 \ldots a_k} \langle q_{k+1}, m_{k+1}, n_{k+1} \rangle$$

  for $a_1, \ldots, a_k \in \Xi \cup \{\varepsilon\}$, $q_i \in Q$, $m_i, n_i \in \mathbb{N}$.

- The language generated by a two-counter machine with output M is

$$L(M) = \{\, w \in \Xi^* \mid \langle q_0, 0, 0 \rangle \xrightarrow{w} \langle q_f, 0, 0 \rangle \,\}$$

  (it accepts only if both counters store 0).

Two-counter Machine with Output Tape

- if $\langle q_1, m_1, n_1 \rangle \xrightarrow{a_1} \langle q_2, m_2, n_2 \rangle \xrightarrow{a_2} \ldots \xrightarrow{a_k} \langle q_{k+1}, m_{k+1}, n_{k+1} \rangle$, we write

$$\langle q_1, m_1, n_1 \rangle \xrightarrow{a_1 \ldots a_k} \langle q_{k+1}, m_{k+1}, n_{k+1} \rangle$$

  for $a_1, \ldots, a_k \in \Xi \cup \{\varepsilon\}$, $q_i \in Q$, $m_i, n_i \in \mathbb{N}$.

- The language generated by a two-counter machine with output M is

$$L(M) = \{\, w \in \Xi^* \mid \langle q_0, 0, 0 \rangle \xrightarrow{w} \langle q_f, 0, 0 \rangle \,\}$$

  (it accepts only if both counters store 0).

- *Any recursively enumerable language is generated by a two-counter machine with output.*

And-branching 2CM without Zerotest with Output

- $M = \langle Q, \Xi, \delta, q_0, q_f \rangle$:
  - Rules:

$$\langle q\ \mathbf{Inc}\ i\ r\rangle, \quad \langle q\ \mathbf{Dec}\ i\ r\rangle, \quad \langle q\ \mathbf{Fork}\ r_1\ r_2\rangle, \quad \langle q\ \mathbf{Write}\ a\ r\rangle$$

for $q, r, r_1, r_2 \in Q,\ i \in \{1, 2\},\ a \in \Xi$.

- A configuration of M is a *sequence* of members of $Q \times \mathbb{N} \times \mathbb{N}$.
  - $X\langle q, m, n\rangle Y \xrightarrow{\varepsilon} X\langle r, m+1, n\rangle Y$ if $\langle q\ \mathbf{Inc}\ 1\ r\rangle \in \delta$,
  - $X\langle q, m+1, n\rangle Y \xrightarrow{\varepsilon} X\langle r, m, n\rangle Y$ if $\langle q\ \mathbf{Dec}\ 1\ r\rangle \in \delta$,

  where $X, Y \in (Q \times \mathbb{N} \times \mathbb{N})^*$.

And-branching 2CM without Zerotest with Output

- $M = \langle Q, \Xi, \delta, q_0, q_f \rangle$:
  - Rules:

$$\langle q\ \mathbf{Inc}\ i\ r\rangle, \quad \langle q\ \mathbf{Dec}\ i\ r\rangle, \quad \langle q\ \mathbf{Fork}\ r_1\ r_2\rangle, \quad \langle q\ \mathbf{Write}\ a\ r\rangle$$

for $q, r, r_1, r_2 \in Q,\ i \in \{1, 2\},\ a \in \Xi$.

- A configuration of M is a *sequence* of members of $Q \times \mathbb{N} \times \mathbb{N}$.
  - $X\langle q, m, n\rangle Y \xrightarrow{\varepsilon} X\langle r, m+1, n\rangle Y$ if $\langle q\ \mathbf{Inc}\ 1\ r\rangle \in \delta$,
  - $X\langle q, m+1, n\rangle Y \xrightarrow{\varepsilon} X\langle r, m, n\rangle Y$ if $\langle q\ \mathbf{Dec}\ 1\ r\rangle \in \delta$,
  - $X\langle q, m, n\rangle Y \xrightarrow{\varepsilon} X\langle r_1, m, n\rangle\langle r_2, m, n\rangle Y$ if $\langle q\ \mathbf{Fork}\ r_1\ r_2\rangle \in \delta$,

where $X, Y \in (Q \times \mathbb{N} \times \mathbb{N})^*$.

And-branching 2CM without Zerotest with Output

- $M = \langle Q, \Xi, \delta, q_0, q_f \rangle$:
  - Rules:

$$\langle q\ \mathbf{Inc}\ i\ r\rangle, \quad \langle q\ \mathbf{Dec}\ i\ r\rangle, \quad \langle q\ \mathbf{Fork}\ r_1\ r_2\rangle, \quad \langle q\ \mathbf{Write}\ a\ r\rangle$$

for $q, r, r_1, r_2 \in Q,\ i \in \{1, 2\},\ a \in \Xi$.

- A configuration of M is a *sequence* of members of $Q \times \mathbb{N} \times \mathbb{N}$.
  - $X\langle q, m, n\rangle Y \xrightarrow{\varepsilon} X\langle r, m+1, n\rangle Y$ if $\langle q\ \mathbf{Inc}\ 1\ r\rangle \in \delta$,
  - $X\langle q, m+1, n\rangle Y \xrightarrow{\varepsilon} X\langle r, m, n\rangle Y$ if $\langle q\ \mathbf{Dec}\ 1\ r\rangle \in \delta$,
  - $X\langle q, m, n\rangle Y \xrightarrow{\varepsilon} X\langle r_1, m, n\rangle\langle r_2, m, n\rangle Y$ if $\langle q\ \mathbf{Fork}\ r_1\ r_2\rangle \in \delta$,
  - $\langle q, m, n\rangle Y \xrightarrow{a} \langle r, m, n\rangle Y$ if $\langle q\ \mathbf{Write}\ a\ r\rangle \in \delta$,
  - $X\langle q, m, n\rangle Y \xrightarrow{\varepsilon} X\langle r, m, n\rangle Y$ if $\langle q\ \mathbf{Write}\ a\ r\rangle \in \delta$ and $X \neq \varepsilon$,

where $X, Y \in (Q \times \mathbb{N} \times \mathbb{N})^*$.

And-branching 2CM without Zerotest with Output

- $M = \langle Q, \Xi, \delta, q_0, q_f \rangle$:
  - Rules:

$$\langle q\ \mathbf{Inc}\ i\ r\rangle, \quad \langle q\ \mathbf{Dec}\ i\ r\rangle, \quad \langle q\ \mathbf{Fork}\ r_1\ r_2\rangle, \quad \langle q\ \mathbf{Write}\ a\ r\rangle$$

    for $q, r, r_1, r_2 \in Q,\ i \in \{1, 2\},\ a \in \Xi$.

- A configuration of M is a *sequence* of members of $Q \times \mathbb{N} \times \mathbb{N}$.
  - $X\langle q, m, n\rangle Y \xrightarrow{\varepsilon} X\langle r, m+1, n\rangle Y$ if $\langle q\ \mathbf{Inc}\ 1\ r\rangle \in \delta$,
  - $X\langle q, m+1, n\rangle Y \xrightarrow{\varepsilon} X\langle r, m, n\rangle Y$ if $\langle q\ \mathbf{Dec}\ 1\ r\rangle \in \delta$,
  - $X\langle q, m, n\rangle Y \xrightarrow{\varepsilon} X\langle r_1, m, n\rangle\langle r_2, m, n\rangle Y$ if $\langle q\ \mathbf{Fork}\ r_1\ r_2\rangle \in \delta$,
  - $\langle q, m, n\rangle Y \xrightarrow{a} \langle r, m, n\rangle Y$ if $\langle q\ \mathbf{Write}\ a\ r\rangle \in \delta$,
  - $X\langle q, m, n\rangle Y \xrightarrow{\varepsilon} X\langle r, m, n\rangle Y$ if $\langle q\ \mathbf{Write}\ a\ r\rangle \in \delta$ and $X \neq \varepsilon$,

  where $X, Y \in (Q \times \mathbb{N} \times \mathbb{N})^*$.

- $L(M) = \{\, w \in \Xi^* \mid \langle q_0, 0, 0\rangle \xrightarrow{w} \langle q_f, 0, 0\rangle \ldots \langle q_f, 0, 0\rangle \,\}$.

And-branching 2CM without Zerotest with Output

- Proposition:
  Any 2CM with output has an equivalent
  and-branching 2CM without zerotest with output.

Encoding by ACG with Cartesian Product

- $M = \langle Q, \Xi, \delta, q_0, q_f \rangle$: And-branching 2CM without zerotest with output
- $\mathscr{G}_M = \langle \Sigma_M, \Sigma_\Xi, \mathscr{L}, q_0 \rangle$: ACG with Cartesian product

Encoding by ACG with Cartesian Product

- $M = \langle Q, \Xi, \delta, q_0, q_f \rangle$: And-branching 2CM without zerotest with output
- $\mathscr{G}_M = \langle \Sigma_M, \Sigma_\Xi, \mathscr{L}, q_0 \rangle$: ACG with Cartesian product
- $A_M = \{p_1, p_2\} \cup Q$
- $\langle q, m, n \rangle \xrightarrow{w} \langle q_f, 0, 0 \rangle \ldots \langle q_f, 0, 0 \rangle$
  iff $\Gamma_{m,n} \vdash_{\Sigma_M} t : q$
  where $\Gamma_{m,n} = x_1 : p_1, \ldots, x_m : p_1,\ y_1 : p_2, \ldots, y_n : p_2$.

Encoding by ACG with Cartesian Product

- $M = \langle Q, \Xi, \delta, q_0, q_f \rangle$: And-branching 2CM without zerotest with output
- $\mathscr{G}_M = \langle \Sigma_M, \Sigma_\Xi, \mathscr{L}, q_0 \rangle$: ACG with Cartesian product

$$\Sigma_M \begin{cases} p_1, p_2 : \mathbf{type}, \\ \quad q : \mathbf{type} \quad \text{for } q \in Q, \\ \hline c_f : q_f, \\ c_\rho : \begin{cases} (p_i \multimap r) \multimap q & \text{for } \rho = \langle q\ \mathbf{Inc}\ i\ r \rangle, \\ r \multimap p_i \multimap q & \text{for } \rho = \langle q\ \mathbf{Dec}\ i\ r \rangle, \\ (r_1 \mathbin{\&} r_2) \multimap q & \text{for } \rho = \langle q\ \mathbf{Fork}\ r_1\ r_2 \rangle, \\ r \multimap q & \text{for } \rho = \langle q\ \mathbf{Write}\ a\ r \rangle. \end{cases} \end{cases}$$

- $\langle q, m, n \rangle \xrightarrow{w} \langle q_f, 0, 0 \rangle \dots \langle q_f, 0, 0 \rangle$
  iff $\Gamma_{m,n} \vdash_{\Sigma_M} t : q$
  where $\Gamma_{m,n} = x_1 : p_1, \dots, x_m : p_1,\ y_1 : p_2, \dots, y_n : p_2$.

Completeness

- $$\begin{cases} c_f : q_f, \\ c_\rho : \begin{cases} (p_i \multimap r) \multimap q & \text{for } \rho = \langle q\ \mathbf{Inc}\ i\ r\rangle, \\ r \multimap p_i \multimap q & \text{for } \rho = \langle q\ \mathbf{Dec}\ i\ r\rangle, \\ (r_1 \mathbin{\&} r_2) \multimap q & \text{for } \rho = \langle q\ \mathbf{Fork}\ r_1\ r_2\rangle, \\ r \multimap q & \text{for } \rho = \langle q\ \mathbf{Write}\ a\ r\rangle. \end{cases} \end{cases}$$
- $\langle q, m, n\rangle \xrightarrow{w} \langle q_f, 0, 0\rangle \dots \langle q_f, 0, 0\rangle \Longrightarrow \Gamma_{m,n} \vdash_{\Sigma_M} t : q.$

Completeness

- $$\begin{cases} c_f : q_f, \\ c_\rho : \begin{cases} (p_i \multimap r) \multimap q & \text{for } \rho = \langle q\ \mathbf{Inc}\ i\ r\rangle, \\ r \multimap p_i \multimap q & \text{for } \rho = \langle q\ \mathbf{Dec}\ i\ r\rangle, \\ (r_1 \,\&\, r_2) \multimap q & \text{for } \rho = \langle q\ \mathbf{Fork}\ r_1\ r_2\rangle, \\ r \multimap q & \text{for } \rho = \langle q\ \mathbf{Write}\ a\ r\rangle. \end{cases} \end{cases}$$
- $\langle q, m, n\rangle \xrightarrow{w} \langle q_f, 0, 0\rangle \ldots \langle q_f, 0, 0\rangle \Longrightarrow \Gamma_{m,n} \vdash_{\Sigma_M} t : q.$
- (Base Case)

$$(\Gamma_{0,0} = \varepsilon) \vdash_{\Sigma_M} c_f : q_f$$

Completeness (Inc)

- $$\begin{cases} c_f : q_f, \\ c_\rho : \begin{cases} (p_i \multimap r) \multimap q & \text{for } \rho = \langle q\ \mathbf{Inc}\ i\ r\rangle, \\ r \multimap p_i \multimap q & \text{for } \rho = \langle q\ \mathbf{Dec}\ i\ r\rangle, \\ (r_1 \,\&\, r_2) \multimap q & \text{for } \rho = \langle q\ \mathbf{Fork}\ r_1\ r_2\rangle, \\ r \multimap q & \text{for } \rho = \langle q\ \mathbf{Write}\ a\ r\rangle. \end{cases} \end{cases}$$
- $\langle q, m, n\rangle \xrightarrow{w} \langle q_f, 0, 0\rangle \ldots \langle q_f, 0, 0\rangle \Longrightarrow \Gamma_{m,n} \vdash_{\Sigma_M} t : q$.
- Suppose $\langle q, m, n\rangle \xrightarrow{\varepsilon} \langle r, m+1, n\rangle \xrightarrow{w} \langle q_f, 0, 0\rangle \ldots \langle q_f, 0, 0\rangle$.
  By induction hypothesis,

$$\frac{\Gamma_{m,n},\ x_{m+1} : p_1 \vdash_{\Sigma_M} t : r}{\qquad\qquad}$$

is provable.

Completeness (Inc)

- $$\begin{cases} c_f : q_f, \\ c_\rho : \begin{cases} (p_i \multimap r) \multimap q & \text{for } \rho = \langle q\ \mathbf{Inc}\ i\ r\rangle, \\ r \multimap p_i \multimap q & \text{for } \rho = \langle q\ \mathbf{Dec}\ i\ r\rangle, \\ (r_1 \,\&\, r_2) \multimap q & \text{for } \rho = \langle q\ \mathbf{Fork}\ r_1\ r_2\rangle, \\ r \multimap q & \text{for } \rho = \langle q\ \mathbf{Write}\ a\ r\rangle. \end{cases} \end{cases}$$
- $\langle q, m, n\rangle \xrightarrow{w} \langle q_f, 0, 0\rangle \ldots \langle q_f, 0, 0\rangle \Longrightarrow \Gamma_{m,n} \vdash_{\Sigma_M} t : q$.
- Suppose $\langle q, m, n\rangle \xrightarrow{\varepsilon} \langle r, m+1, n\rangle \xrightarrow{w} \langle q_f, 0, 0\rangle \ldots \langle q_f, 0, 0\rangle$. Then,

$$\frac{\Gamma_{m,n},\ x_{m+1} : p_1 \vdash_{\Sigma_M} t : r}{\Gamma_{m,n} \vdash \lambda^\circ x_{m+1}.t : p_1 \multimap r}$$

is provable.

Completeness (Inc)

- $$\begin{cases} c_f : q_f, \\ c_\rho : \begin{cases} (p_i \multimap r) \multimap q & \text{for } \rho = \langle q\ \mathbf{Inc}\ i\ r\rangle, \\ r \multimap p_i \multimap q & \text{for } \rho = \langle q\ \mathbf{Dec}\ i\ r\rangle, \\ (r_1 \,\&\, r_2) \multimap q & \text{for } \rho = \langle q\ \mathbf{Fork}\ r_1\ r_2\rangle, \\ r \multimap q & \text{for } \rho = \langle q\ \mathbf{Write}\ a\ r\rangle. \end{cases} \end{cases}$$
- $\langle q, m, n\rangle \xrightarrow{w} \langle q_f, 0, 0\rangle \ldots \langle q_f, 0, 0\rangle \Longrightarrow \Gamma_{m,n} \vdash_{\Sigma_M} t : q$.
- Suppose $\langle q, m, n\rangle \xrightarrow{\varepsilon} \langle r, m+1, n\rangle \xrightarrow{w} \langle q_f, 0, 0\rangle \ldots \langle q_f, 0, 0\rangle$. Then,

$$\frac{\vdash_{\Sigma_M} c_\rho : (p_1 \multimap r) \multimap q \qquad \dfrac{\Gamma_{m,n},\ x_{m+1} : p_1 \vdash_{\Sigma_M} t : r}{\Gamma_{m,n} \vdash \lambda^\circ x_{m+1}.t : p_1 \multimap r}}{}$$

is provable.

Completeness (Inc)

- $$\begin{cases} c_f : q_f, \\ c_\rho : \begin{cases} (p_i \multimap r) \multimap q & \text{for } \rho = \langle q\ \mathbf{Inc}\ i\ r\rangle, \\ r \multimap p_i \multimap q & \text{for } \rho = \langle q\ \mathbf{Dec}\ i\ r\rangle, \\ (r_1 \,\&\, r_2) \multimap q & \text{for } \rho = \langle q\ \mathbf{Fork}\ r_1\ r_2\rangle, \\ r \multimap q & \text{for } \rho = \langle q\ \mathbf{Write}\ a\ r\rangle. \end{cases} \end{cases}$$
- $\langle q, m, n\rangle \xrightarrow{w} \langle q_f, 0, 0\rangle \ldots \langle q_f, 0, 0\rangle \Longrightarrow \Gamma_{m,n} \vdash_{\Sigma_M} t : q$.
- Suppose $\langle q, m, n\rangle \xrightarrow{\varepsilon} \langle r, m+1, n\rangle \xrightarrow{w} \langle q_f, 0, 0\rangle \ldots \langle q_f, 0, 0\rangle$. Then,

$$\frac{\vdash_{\Sigma_M} c_\rho : (p_1 \multimap r) \multimap q \quad \dfrac{\Gamma_{m,n},\ x_{m+1} : p_1 \vdash_{\Sigma_M} t : r}{\Gamma_{m,n} \vdash \lambda^\circ x_{m+1}.t : p_1 \multimap r}}{\Gamma_{m,n} \vdash_{\Sigma_M} c_\rho(\lambda^\circ x_{m+1}.t) : q}$$

is provable.

Completeness (Fork)

- $$\begin{cases} c_f : q_f, \\ c_\rho : \begin{cases} (p_i \multimap r) \multimap q & \text{for } \rho = \langle q\ \mathbf{Inc}\ i\ r\rangle, \\ r \multimap p_i \multimap q & \text{for } \rho = \langle q\ \mathbf{Dec}\ i\ r\rangle, \\ (r_1\ \&\ r_2) \multimap q & \text{for } \rho = \langle q\ \mathbf{Fork}\ r_1\ r_2\rangle, \\ r \multimap q & \text{for } \rho = \langle q\ \mathbf{Write}\ a\ r\rangle. \end{cases} \end{cases}$$
- $\langle q, m, n\rangle \xrightarrow{w} \langle q_f, 0, 0\rangle \ldots \langle q_f, 0, 0\rangle \Longrightarrow \Gamma_{m,n} \vdash_{\Sigma_M} t : q.$
- $\langle q, m, n\rangle \xrightarrow{\varepsilon} \langle r_1, m, n\rangle\langle r_2, m, n\rangle \xrightarrow{w} \langle q_f, 0, 0\rangle \ldots \langle q_f, 0, 0\rangle.$
  By induction hypothesis,

$$\frac{\Gamma_{m,n} \vdash_{\Sigma_M} t_1 : r_1 \quad \Gamma_{m,n} \vdash_{\Sigma_M} t_2 : r_2}{\qquad\qquad}$$

is provable.

Completeness (Fork)

- $$\begin{cases} c_f : q_f, \\ c_\rho : \begin{cases} (p_i \multimap r) \multimap q & \text{for } \rho = \langle q\ \mathbf{Inc}\ i\ r\rangle, \\ r \multimap p_i \multimap q & \text{for } \rho = \langle q\ \mathbf{Dec}\ i\ r\rangle, \\ (r_1\ \&\ r_2) \multimap q & \text{for } \rho = \langle q\ \mathbf{Fork}\ r_1\ r_2\rangle, \\ r \multimap q & \text{for } \rho = \langle q\ \mathbf{Write}\ a\ r\rangle. \end{cases} \end{cases}$$
- $\langle q, m, n\rangle \xrightarrow{w} \langle q_f, 0, 0\rangle \ldots \langle q_f, 0, 0\rangle \Longrightarrow \Gamma_{m,n} \vdash_{\Sigma_M} t : q.$
- $\langle q, m, n\rangle \xrightarrow{\varepsilon} \langle r_1, m, n\rangle\langle r_2, m, n\rangle \xrightarrow{w} \langle q_f, 0, 0\rangle \ldots \langle q_f, 0, 0\rangle.$
  Then,

$$\frac{\Gamma_{m,n} \vdash_{\Sigma_M} t_1 : r_1 \quad \Gamma_{m,n} \vdash_{\Sigma_M} t_2 : r_2}{\Gamma_{m,n} \vdash_{\Sigma_M} \langle t_1, t_2\rangle : r_1\ \&\ r_2}$$

is provable.

Completeness (Fork)

- $$\begin{cases} c_f : q_f, \\ c_\rho : \begin{cases} (p_i \multimap r) \multimap q & \text{for } \rho = \langle q\ \mathrm{Inc}\ i\ r\rangle, \\ r \multimap p_i \multimap q & \text{for } \rho = \langle q\ \mathrm{Dec}\ i\ r\rangle, \\ (r_1\ \&\ r_2) \multimap q & \text{for } \rho = \langle q\ \mathbf{Fork}\ r_1\ r_2\rangle, \\ r \multimap q & \text{for } \rho = \langle q\ \mathrm{Write}\ a\ r\rangle. \end{cases} \end{cases}$$
- $\langle q, m, n\rangle \xrightarrow{w} \langle q_f, 0, 0\rangle \dots \langle q_f, 0, 0\rangle \Longrightarrow \Gamma_{m,n} \vdash_{\Sigma_M} t : q.$
- $\langle q, m, n\rangle \xrightarrow{\varepsilon} \langle r_1, m, n\rangle\langle r_2, m, n\rangle \xrightarrow{w} \langle q_f, 0, 0\rangle \dots \langle q_f, 0, 0\rangle.$
  Then,

$$\vdash_{\Sigma_M} c_\rho : (r_1\ \&\ r_2) \multimap q \qquad \frac{\Gamma_{m,n} \vdash_{\Sigma_M} t_1 : r_1 \quad \Gamma_{m,n} \vdash_{\Sigma_M} t_2 : r_2}{\Gamma_{m,n} \vdash_{\Sigma_M} \langle t_1, t_2\rangle : r_1\ \&\ r_2}$$

is provable.

Completeness (Fork)

- $$\begin{cases} c_f : q_f, \\ c_\rho : \begin{cases} (p_i \multimap r) \multimap q & \text{for } \rho = \langle q\ \mathbf{Inc}\ i\ r\rangle, \\ r \multimap p_i \multimap q & \text{for } \rho = \langle q\ \mathbf{Dec}\ i\ r\rangle, \\ (r_1\ \&\ r_2) \multimap q & \text{for } \rho = \langle q\ \mathbf{Fork}\ r_1\ r_2\rangle, \\ r \multimap q & \text{for } \rho = \langle q\ \mathbf{Write}\ a\ r\rangle. \end{cases} \end{cases}$$
- $\langle q, m, n\rangle \xrightarrow{w} \langle q_f, 0, 0\rangle \ldots \langle q_f, 0, 0\rangle \Longrightarrow \Gamma_{m,n} \vdash_{\Sigma_M} t : q.$
- $\langle q, m, n\rangle \xrightarrow{\varepsilon} \langle r_1, m, n\rangle\langle r_2, m, n\rangle \xrightarrow{w} \langle q_f, 0, 0\rangle \ldots \langle q_f, 0, 0\rangle.$
  Then,

$$\dfrac{\vdash_{\Sigma_M} c_\rho : (r_1\ \&\ r_2) \multimap q \qquad \dfrac{\Gamma_{m,n} \vdash_{\Sigma_M} t_1 : r_1 \quad \Gamma_{m,n} \vdash_{\Sigma_M} t_2 : r_2}{\Gamma_{m,n} \vdash_{\Sigma_M} \langle t_1, t_2\rangle : r_1\ \&\ r_2}}{\Gamma_{m,n} \vdash_{\Sigma_M} c_\rho\langle t_1.t_2\rangle : q}$$

is provable.

Correctness

- $$\begin{cases} c_f : q_f, \\ c_\rho : \begin{cases} (p_i \multimap r) \multimap q & \text{for } \rho = \langle q\ \mathbf{Inc}\ i\ r\rangle, \\ r \multimap p_i \multimap q & \text{for } \rho = \langle q\ \mathbf{Dec}\ i\ r\rangle, \\ (r_1 \mathbin{\&} r_2) \multimap q & \text{for } \rho = \langle q\ \mathbf{Fork}\ r_1\ r_2\rangle, \\ r \multimap q & \text{for } \rho = \langle q\ \mathbf{Write}\ a\ r\rangle. \end{cases} \end{cases}$$
- $\langle q, m, n\rangle \xrightarrow{w} \langle q_f, 0, 0\rangle \dots \langle q_f, 0, 0\rangle \iff \Gamma_{m,n} \vdash_{\Sigma_M} t : q.$

Lexicon

$c \in C_M$	$\tau_M(c)$	$\mathscr{L}(c)$	
c_f	q_f	$\lambda^\circ z.z$	
c_ρ	$(p_i \multimap r) \multimap q$	$\lambda^\circ x.x(\lambda^\circ z.z)$	$\rho = \langle q\ \mathbf{Inc}\ i\ r\rangle$
	$r \multimap p_i \multimap q$	$\lambda^\circ xyz.x(yz)$	$\rho = \langle q\ \mathbf{Dec}\ i\ r\rangle$
	$(r_1\ \&\ r_2) \multimap q$	$\lambda^\circ x.\pi_1 x$	$\rho = \langle q\ \mathbf{Fork}\ r_1\ r_2\rangle$
	$r \multimap q$	$\lambda^\circ x./a/ + x$	$\rho = \langle q\ \mathbf{Write}\ a\ r\rangle$

- $\langle q, m, n\rangle \xrightarrow{w} \langle q_f, 0, 0\rangle \ldots \langle q_f, 0, 0\rangle$

  iff $\begin{cases} \Gamma_{m,n} \vdash_{\Sigma_M} t : q & \text{and} \\ \mathscr{L}(t)[z := /\varepsilon/]_{z \in \mathsf{FV}(t)} = /w/. \end{cases}$

$$L(M) = \mathcal{O}(\mathscr{G}_M)$$

Theorem

- For any two-counter machine with output tape,
  one can construct a third-order ACG with Cartesian product
  that generates the same language.

Theorem

- For any two-counter machine with output tape, one can construct a third-order ACG with Cartesian product that generates the same language.

$$order(\alpha \,\&\, \beta) = \max\{order(\alpha), order(\beta)\}$$

$$order(\alpha \multimap \beta) = \max\{order(\alpha) + 1, order(\beta)\}$$

5. Conclusion

Conclusion

- Any recursively enumerable language is generated by a second-order ACG with dependent product.

- Any recursively enumerable language is generated by a third-order ACG with Cartesian product.

Conclusion

- Any recursively enumerable language is generated by a second-order ACG with dependent product.
  - Any first-order ACGs with dependent product generates a finite language.
- Any recursively enumerable language is generated by a third-order ACG with Cartesian product.

Conclusion

- Any recursively enumerable language is generated by a second-order ACG with dependent product.
  - Any first-order ACGs with dependent product generates a finite language.
- Any recursively enumerable language is generated by a third-order ACG with Cartesian product.
  - The generative capacity of second-order ACGs with Cartesian product coincides with that of second-order (linear implicative) ACGs modulo non simply typed terms.

Conclusion

- Any recursively enumerable language is generated by a second-order ACG with dependent product.
  - Any first-order ACGs with dependent product generates a finite language.
- Any recursively enumerable language is generated by a third-order ACG with Cartesian product.
  - The generative capacity of second-order ACGs with Cartesian product coincides with that of second-order (linear implicative) ACGs modulo non simply typed terms.

$\Longrightarrow$ Finding appropriate fragments of extensions

$\left\{\begin{array}{l}\text{Expressive power}\\ \text{Tractability}\end{array}\right.$