Parsing Abstract Categorial Grammars: Complexity and Algorithms

Sylvain Salvati

INPL-INRIA

Introduction

Parsing Abstract Categorial Grammars ([de Groote-2001]):

- The general case
- Natural language case
- Grammars whose abstract language is regular ($\mathcal{L}(2, m)$)

Preliminaries

Given an ACG $\mathcal{G} = \langle \Sigma_1, \Sigma_2, \mathcal{L}, S \rangle$

- $\mathcal{G}$ is lexicalized iff for all abstract constant c, $\mathcal{L}(c)$ contains at least an object constant
- $\mathcal{G}$ is semi-lexicalized iff all abstract constant c either has a second order type or $\mathcal{L}(c)$ contains at least an object constant

The general case

Given an ACG $\mathcal{G} = \langle \Sigma_1, \Sigma_2, \mathcal{L}, S \rangle$ and a term u, question is whether there is an abstract term t of type S so that $\mathcal{L}(t) =_{\beta\eta} u$.

The general case

Given an ACG $\mathcal{G} = \langle \Sigma_1, \Sigma_2, \mathcal{L}, S \rangle$ and a term u, question is whether there is an abstract term t of type S so that $\mathcal{L}(t) =_{\beta\eta} u$.

Algorithm principle: try to incrementaly build the abstract term in a top down approach.

The general case

Given an ACG $\mathcal{G} = \langle \Sigma_1, \Sigma_2, \mathcal{L}, S \rangle$ and a term u, question is whether there is an abstract term t of type S so that $\mathcal{L}(t) =_{\beta\eta} u$.

Algorithm principle: try to incrementaly build the abstract term in a top down approach.

Agorithm items: $\langle \Gamma; v; \alpha \rangle$

The general case

Given an ACG $\mathcal{G} = \langle \Sigma_1, \Sigma_2, \mathcal{L}, S \rangle$ and a term u, question is whether there is an abstract term t of type S so that $\mathcal{L}(t) =_{\beta\eta} u$.
Algorithm principle: try to incrementaly build the abstract term in a top down approach.
Agorithm items: $\langle \Gamma; v; \alpha \rangle$

- Γ : context which associates abstract types to variables

The general case

Given an ACG $\mathcal{G} = \langle \Sigma_1, \Sigma_2, \mathcal{L}, S \rangle$ and a term u, question is whether there is an abstract term t of type S so that $\mathcal{L}(t) =_{\beta\eta} u$.
Algorithm principle: try to incrementaly build the abstract term in a top down approach.
Agorithm items: $\langle \Gamma; v; \alpha \rangle$

- Γ : context which associates abstract types to variables
- v : object term in β-normal η-long form

The general case

Given an ACG $\mathcal{G} = \langle \Sigma_1, \Sigma_2, \mathcal{L}, S \rangle$ and a term u, question is whether there is an abstract term t of type S so that $\mathcal{L}(t) =_{\beta\eta} u$.

Algorithm principle: try to incrementaly build the abstract term in a top down approach.

Agorithm items: $\langle \Gamma; v; \alpha \rangle$

- Γ : context which associates abstract types to variables
- v : object term in β-normal η-long form
- α : abstract type

The general case

Given an ACG $\mathcal{G} = \langle \Sigma_1, \Sigma_2, \mathcal{L}, S \rangle$ and a term u, question is whether there is an abstract term t of type S so that $\mathcal{L}(t) =_{\beta\eta} u$.
Algorithm principle: try to incrementaly build the abstract term in a top down approach.
Agorithm items: $\langle \Gamma; v; \alpha \rangle$

- Γ : context which associates abstract types to variables
- v : object term in β-normal η-long form
- α : abstract type

the item $\langle \Gamma; v; \alpha \rangle$ represents the problem of finding an abstract term t such that:

$$\Gamma \vdash t : \alpha \text{ and } \mathcal{L}(t) =_{\beta\eta} v$$

Rules of General Algorithm

The algorithm is described by a rewriting system on sets of items:

$$\mathcal{I} \rightarrow_A (\mathcal{I} \backslash I) \cup \mathcal{J} \text{ if } I \rightarrow_a \mathcal{J}$$

Rules of General Algorithm

The algorithm is described by a rewriting system on sets of items:

$$\mathcal{I} \rightarrow_A (\mathcal{I} \backslash I) \cup \mathcal{J} \text{ if } I \rightarrow_a \mathcal{J}$$

with:

- $\langle \Gamma; \lambda x.v; \alpha \multimap \beta \rangle \rightarrow_a \{\langle \Gamma, x : \alpha; v; \beta \rangle\}$

Rules of General Algorithm

The algorithm is described by a rewriting system on sets of items:

$$\mathcal{I} \rightarrow_A (\mathcal{I} \backslash I) \cup \mathcal{J} \text{ if } I \rightarrow_a \mathcal{J}$$

with:

- $\langle \Gamma; \lambda x.v; \alpha \multimap \beta \rangle \rightarrow_a \{\langle \Gamma, x : \alpha; v; \beta \rangle\}$
- $\langle \Gamma, x : \alpha_1 \multimap \ldots \multimap \alpha_n \multimap A; \overline{xv_1 \ldots v_n}; A \rangle \rightarrow_a \{\langle \Gamma_1; \overline{v_1}; \alpha_1 \rangle; \ldots; \langle \Gamma_n; \overline{v_n}; \alpha_n \rangle\}$

Rules of General Algorithm

The algorithm is described by a rewriting system on sets of items:

$$\mathcal{I} \rightarrow_A (\mathcal{I} \backslash I) \cup \mathcal{J} \text{ if } I \rightarrow_a \mathcal{J}$$

with:

- $\langle \Gamma; \lambda x.v; \alpha \multimap \beta \rangle \rightarrow_a \{\langle \Gamma, x : \alpha; v; \beta \rangle\}$
- $\langle \Gamma, x : \alpha_1 \multimap \ldots \multimap \alpha_n \multimap A; \overline{x v_1 \ldots v_n}; A \rangle \rightarrow_a \{\langle \Gamma_1; \overline{v_1}; \alpha_1 \rangle; \ldots; \langle \Gamma_n; \overline{v_n}; \alpha_n \rangle\}$
- $\langle \Gamma, v, A \rangle \rightarrow_a \{\langle \Gamma_1; t_1; \alpha_1 \rangle; \ldots; \langle \Gamma_n; t_n; \alpha_n \rangle\}$ if there is an abstract constant c of type $\alpha_1 \multimap \ldots \multimap \alpha_n \multimap A$ so that $\mathcal{L}(c) t_1 \ldots t_n =_{\beta\eta} v$

Rules of General Algorithm

The algorithm is described by a rewriting system on sets of items:

$$\mathcal{I} \rightarrow_A (\mathcal{I} \backslash I) \cup \mathcal{J} \text{ if } I \rightarrow_a \mathcal{J}$$

with:

- $\langle \Gamma; \lambda x.v; \alpha \multimap \beta \rangle \rightarrow_a \{\langle \Gamma, x : \alpha; v; \beta \rangle\}$
- $\langle \Gamma, x : \alpha_1 \multimap \ldots \multimap \alpha_n \multimap A; \overline{x v_1 \ldots v_n}; A \rangle \rightarrow_a \{\langle \Gamma_1; \overline{v_1}; \alpha_1 \rangle; \ldots ; \langle \Gamma_n; \overline{v_n}; \alpha_n \rangle\}$
- $\langle \Gamma, v, A \rangle \rightarrow_a \{\langle \Gamma_1; t_1; \alpha_1 \rangle; \ldots ; \langle \Gamma_n; t_n; \alpha_n \rangle\}$ if there is an abstract constant c of type $\alpha_1 \multimap \ldots \multimap \alpha_n \multimap A$ so that $\mathcal{L}(c) t_1 \ldots t_n =_{\beta\eta} v$

To make the last rule work we need solve matching equations (NP-complete)

Properties of the Algorithm

- $\{\langle \cdot; u; S\rangle\} \xrightarrow{*}_A \emptyset$ if and only if there is an abstract term t of type S such that $\mathcal{L}(t) =_{\beta\eta} u$

Properties of the Algorithm

- $\{\langle \cdot ; u ; S \rangle\} \xrightarrow{*}_A \emptyset$ if and only if there is an abstract term t of type S such that $\mathcal{L}(t) =_{\beta\eta} u$
- termination: No

Properties of the Algorithm

- $\{\langle \cdot; u; S \rangle\} \xrightarrow{*}_A \emptyset$ if and only if there is an abstract term t of type S such that $\mathcal{L}(t) =_{\beta\eta} u$
- termination: No
- termination in the lexicalized case : OK

Properties of the Algorithm

- $\{\langle \cdot ; u; S\rangle\} \xrightarrow{*}_A \emptyset$ if and only if there is an abstract term t of type S such that $\mathcal{L}(t) =_{\beta\eta} u$
- termination: No
- termination in the lexicalized case : OK
  - the algorithm is in that case Non-deterministic Polynomial

Properties of the Algorithm

- $\{\langle \cdot ; u; S\rangle\} \xrightarrow{*}_A \emptyset$ if and only if there is an abstract term t of type S such that $\mathcal{L}(t) =_{\beta\eta} u$
- termination: No
- termination in the lexicalized case : OK
  - the algorithm is in that case Non-deterministic Polynomial
  - membership problem is polynomial for LACGs of $\mathcal{L}(2, m)$ but this algorithm can have an exponential behaviour.

Properties of the Algorithm

- $\{\langle \cdot ; u ; S \rangle\} \xrightarrow{*}_A \emptyset$ if and only if there is an abstract term t of type S such that $\mathcal{L}(t) =_{\beta\eta} u$
- termination: No
- termination in the lexicalized case : OK
  - the algorithm is in that case Non-deterministic Polynomial
  - membership problem is polynomial for LACGs of $\mathcal{L}(2, m)$ but this algorithm can have an exponential behaviour.
- if we tabulate intermediate results:

Properties of the Algorithm

- $\{\langle\cdot; u; S\rangle\} \xrightarrow{*}_A \emptyset$ if and only if there is an abstract term t of type S such that $\mathcal{L}(t) =_{\beta\eta} u$
- termination: No
- termination in the lexicalized case : OK
  - the algorithm is in that case Non-deterministic Polynomial
  - membership problem is polynomial for LACGs of $\mathcal{L}(2, m)$ but this algorithm can have an exponential behaviour.
- if we tabulate intermediate results:
  - termination: unknown (parsing ACG is at least as hard as proving in MELL)

Properties of the Algorithm

- $\{\langle\cdot; u; S\rangle\} \xrightarrow{*}_A \emptyset$ if and only if there is an abstract term t of type S such that $\mathcal{L}(t) =_{\beta\eta} u$
- termination: No
- termination in the lexicalized case : OK
  - the algorithm is in that case Non-deterministic Polynomial
  - membership problem is polynomial for LACGs of $\mathcal{L}(2, m)$ but this algorithm can have an exponential behaviour.
- if we tabulate intermediate results:
  - termination: unknown (parsing ACG is at least as hard as proving in MELL)
  - termination in the semi-lexicalized case : OK

Properties of the Algorithm

- $\{\langle \cdot ; u ; S \rangle\} \xrightarrow{*}_A \emptyset$ if and only if there is an abstract term t of type S such that $\mathcal{L}(t) =_{\beta\eta} u$
- termination: No
- termination in the lexicalized case : OK
  - the algorithm is in that case Non-deterministic Polynomial
  - membership problem is polynomial for LACGs of $\mathcal{L}(2, m)$ but this algorithm can have an exponential behaviour.
- if we tabulate intermediate results:
  - termination: unknown (parsing ACG is at least as hard as proving in MELL)
  - termination in the semi-lexicalized case : OK
    - exponential even for membership problem which is in NP

Properties of the Algorithm

- $\{\langle\cdot; u; S\rangle\} \xrightarrow{*}_A \emptyset$ if and only if there is an abstract term t of type S such that $\mathcal{L}(t) =_{\beta\eta} u$
- termination: No
- termination in the lexicalized case : OK
  - the algorithm is in that case Non-deterministic Polynomial
  - membership problem is polynomial for LACGs of $\mathcal{L}(2, m)$ but this algorithm can have an exponential behaviour.
- if we tabulate intermediate results:
  - termination: unknown (parsing ACG is at least as hard as proving in MELL)
  - termination in the semi-lexicalized case : OK
    - exponential even for membership problem which is in NP
    - exponential in the size of the grammar for ACGs coding CFGs

Incremental algorithm

- This algorithm is dedicated to the analysis of lexicalized ACGs describing a natural language.

Incremental algorithm

- This algorithm is dedicated to the analysis of lexicalized ACGs describing a natural language.
- Based on an approach proposed by Morrill to parse in Lambek's Calculus [Morrill-2000] and on a way to build proof-nets proposed by de Groote [de Groote-2000].

Incremental algorithm

- This algorithm is dedicated to the analysis of lexicalized ACGs describing a natural language.
- Based on an approach proposed by Morrill to parse in Lambek's Calculus [Morrill-2000] and on a way to build proof-nets proposed by de Groote [de Groote-2000].
- It handles polarized items of the form $\langle t, u, A^{\epsilon} \rangle$ where $\epsilon \in \{+; -; \circ\}$

Incremental algorithm

The use of this algorithm necessitates to prepare the lexicon. Each abstract constant c will be represented by a collection of items $I_c = It(c, \tau(c)^-)$ where:

- $It(t, (\alpha \multimap \beta)^+) = It(x, \alpha^-) \cup It(tx, \beta^+)$ where x is a fresh variable

Incremental algorithm

The use of this algorithm necessitates to prepare the lexicon. Each abstract constant c will be represented by a collection of items $I_c = It(c, \tau(c)^-)$ where:

- $It(t, (\alpha \multimap \beta)^+) = It(x, \alpha^-) \cup It(tx, \beta^+)$ where x is a fresh variable
- $It(t, (\alpha \multimap \beta)^-) = It(\mathbf{X}, \alpha^+) \cup It(t\mathbf{X}, \beta^-)$ where $\mathbf{X}$ is a fresh unknown

Incremental algorithm

The use of this algorithm necessitates to prepare the lexicon. Each abstract constant c will be represented by a collection of items $I_c = It(c, \tau(c)^-)$ where:

- $It(t, (\alpha \multimap \beta)^+) = It(x, \alpha^-) \cup It(tx, \beta^+)$ where x is a fresh variable
- $It(t, (\alpha \multimap \beta)^-) = It(\mathbf{X}, \alpha^+) \cup It(t\mathbf{X}, \beta^-)$ where $\mathbf{X}$ is a fresh unknown
- $It(t, A^+) = \langle t, \mathcal{L}(t), A^+ \rangle$ and $It(t, A^-) = \langle t, \mathcal{L}(t), A^- \rangle$

Incremental algorithm

The use of this algorithm necessitates to prepare the lexicon. Each abstract constant c will be represented by a collection of items $I_c = It(c, \tau(c)^-)$ where:

- $It(t, (\alpha \multimap \beta)^+) = It(x, \alpha^-) \cup It(tx, \beta^+)$ where x is a fresh variable
- $It(t, (\alpha \multimap \beta)^-) = It(\mathbf{X}, \alpha^+) \cup It(t\mathbf{X}, \beta^-)$ where $\mathbf{X}$ is a fresh unknown
- $It(t, A^+) = \langle t, \mathcal{L}(t), A^+ \rangle$ and $It(t, A^-) = \langle t, \mathcal{L}(t), A^- \rangle$

Note: each time we use the set of items I_c, we require freshness of the unknowns and the variables it uses.

Handeling items

$\langle t, \mathcal{L}(t), A^+ \rangle$ and $\langle v, \mathcal{L}(v), A^- \rangle$ are complementary if:

- $t = \mathbf{X} x_1 \ldots x_n$ and $\{x_1; \ldots; x_n\} \subseteq FV(v)$
- $\lambda x_1 \ldots x_n.v$ is their *unifier* and $\mathbf{X}$ is their *unification support*

Handling object constants

- When we parse strings we put the constants it contains into a list L. If $u = \lambda x.a_1(a_2(\ldots(a_n x)\ldots))$ $L_u = [a_1; \ldots; a_n]$.

Handling object constants

- When we parse strings we put the constants it contains into a list L. If $u = \lambda x.a_1(a_2(\ldots(a_n x)\ldots))$ $L_u = [a_1; \ldots; a_n]$.
- Given an abstract constant c we associate the list of the constants it contains to be L_c.

Handling object constants

- When we parse strings we put the constants it contains into a list L. If $u = \lambda x.a_1(a_2(\ldots(a_n x)\ldots))$ $L_u = [a_1; \ldots; a_n]$.
- Given an abstract constant c we associate the list of the constants it contains to be L_c.
- We note $L' \subseteq L$ is the multiset of constants of L' is included in the one of L.

Handling object constants

- When we parse strings we put the constants it contains into a list L. If $u = \lambda x.a_1(a_2(\ldots(a_n x)\ldots))$ $L_u = [a_1; \ldots; a_n]$.
- Given an abstract constant c we associate the list of the constants it contains to be L_c.
- We note $L' \subseteq L$ is the multiset of constants of L' is included in the one of L.
- $L' \sqsubseteq L$ if $L' \subseteq L$ and the first elements of L is in L'.

Handling object constants

- When we parse strings we put the constants it contains into a list L. If $u = \lambda x.a_1(a_2(\ldots(a_n x)\ldots))$ $L_u = [a_1; \ldots; a_n]$.
- Given an abstract constant c we associate the list of the constants it contains to be L_c.
- We note $L' \subseteq L$ is the multiset of constants of L' is included in the one of L.
- $L' \sqsubseteq L$ if $L' \subseteq L$ and the first elements of L is in L'.
- If $L' \subseteq L$ then $L \backslash L'$ is the list where we suppress the top-most elements of L' which are present in L.

Rules of the algorithm

The algorithm acts on triples $(\mathcal{I}, L, u)$ where $\mathcal{I}$ is a set of items and L is a list of object constants.

Rules of the algorithm

The algorithm acts on triples $(\mathcal{I}, L, u)$ where $\mathcal{I}$ is a set of items and L is a list of object constants.
It uses the following rules:

- $(\mathcal{I}, L) \rightarrow_{Inc} (\mathcal{I} \cup I_c, L \backslash L_c)$ if $L_c \sqsubseteq L$

Rules of the algorithm

The algorithm acts on triples $(\mathcal{I}, L, u)$ where $\mathcal{I}$ is a set of items and L is a list of object constants.
It uses the following rules:

- $(\mathcal{I}, L) \rightarrow_{Inc} (\mathcal{I} \cup I_c, L \backslash L_c)$ if $L_c \sqsubseteq L$
- $(\mathcal{I} \cup \{I_1; I_2\}, L) \rightarrow_{Inc} (\mathcal{I}[\mathbf{X} := v] \cup \{\langle v, \mathcal{L}(v), A^\circ \rangle\}, L)$ if I_1 and I_2 are complementary and v is their unifier and $\mathbf{X}$ their unification support

Rules of the algorithm

The algorithm acts on triples $(\mathcal{I}, L, u)$ where $\mathcal{I}$ is a set of items and L is a list of object constants.
It uses the following rules:

- $(\mathcal{I}, L) \rightarrow_{Inc} (\mathcal{I} \cup I_c, L \backslash L_c)$ if $L_c \sqsubseteq L$
- $(\mathcal{I} \cup \{I_1; I_2\}, L) \rightarrow_{Inc} (\mathcal{I}[\mathbf{X} := v] \cup \{\langle v, \mathcal{L}(v), A^\circ \rangle\}, L)$ if I_1 and I_2 are complementary and v is their unifier and $\mathbf{X}$ their unification support

$(\langle \mathbf{X}, \mathbf{X}, S^+ \rangle, L_u, u) \xrightarrow{*}_{Inc} (\mathcal{I}, [], u)$ with $\langle t, u, S^\circ \rangle \in \mathcal{I}$ iff t is an abstract term of type S and $\mathcal{L}(t) =_{\beta\eta} u$.

Example

- $J := \lambda z.|\mathrm{John}|\, z : NP$
- $M := \lambda z.|\mathrm{Mary}|\, z : NP$
- $L := \lambda xyz.x(|\mathrm{loves}|(y\, z))) : NP \multimap NP \multimap S$

Example

- $\langle J, \lambda z.|\mathrm{John}|\, z, NP^{-}\rangle$
- $M := \lambda z.|\mathrm{Mary}|\, z : NP$
- $L := \lambda xyz.x(|\mathrm{loves}|(y\, z))) : NP \multimap NP \multimap S$

Example

- $\langle J, \lambda z.|\mathrm{John}|\, z, NP^{-}\rangle$
- $\langle M, \lambda z.|\mathrm{Mary}|\, z, NP^{-}\rangle$
- $L := \lambda xyz.x(|\mathrm{loves}|(y\, z))) : NP \multimap NP \multimap S$

Example

- $\langle J, \lambda z.|\mathrm{John}|\, z, NP^{-}\rangle$
- $\langle M, \lambda z.|\mathrm{Mary}|\, z, NP^{-}\rangle$
- $\langle L\,\mathbf{Y}\,\mathbf{Z}, \lambda z.\mathbf{Y}(|\mathrm{loves}|(\mathbf{Z}\,z)), S^{-}\rangle$, $\langle \mathbf{Y}, \mathbf{Y}, NP^{+}\rangle$, $\langle \mathbf{Z}, \mathbf{Z}, NP^{+}\rangle$

Example

- $\langle J, \lambda z.|\mathrm{John}|\, z, NP^{-}\rangle$
- $\langle M, \lambda z.|\mathrm{Mary}|\, z, NP^{-}\rangle$
- $\langle L\, \mathbf{Y}\, \mathbf{Z}, \lambda z.\mathbf{Y}(|\mathrm{loves}|(\mathbf{Z}\, z)), S^{-}\rangle$, $\langle \mathbf{Y}, \mathbf{Y}, NP^{+}\rangle$, $\langle \mathbf{Z}, \mathbf{Z}, NP^{+}\rangle$

$\lambda z.|\mathrm{John}|(|\mathrm{loves}|(|\mathrm{Mary}|\, z))$

$\langle \mathbf{X}, \mathbf{X}, S^{+}\rangle$

$$\begin{vmatrix} |\mathrm{John}| \\ |\mathrm{loves}| \\ |\mathrm{Mary}| \\ \\ \end{vmatrix}$$

Example

- $\langle J, \lambda z.|\text{John}|\, z, NP^{-}\rangle$
- $\langle M, \lambda z.|\text{Mary}|\, z, NP^{-}\rangle$
- $\langle L\,\mathbf{Y}\,\mathbf{Z}, \lambda z.\mathbf{Y}(|\text{loves}|(\mathbf{Z}\,z)), S^{-}\rangle$, $\langle \mathbf{Y}, \mathbf{Y}, NP^{+}\rangle$, $\langle \mathbf{Z}, \mathbf{Z}, NP^{+}\rangle$

$\lambda z.|\text{John}|(|\text{loves}|(|\text{Mary}|\, z))$

$\langle \mathbf{X}, \mathbf{X}, S^{+}\rangle$

|John|
|loves|
|Mary|

Example

- $\langle J, \lambda z.|\mathrm{John}|\, z, NP^- \rangle$
- $\langle M, \lambda z.|\mathrm{Mary}|\, z, NP^- \rangle$
- $\langle L\,\mathbf{Y}\,\mathbf{Z}, \lambda z.\mathbf{Y}(|\mathrm{loves}|(\mathbf{Z}\,z)), S^- \rangle$, $\langle \mathbf{Y}, \mathbf{Y}, NP^+ \rangle$, $\langle \mathbf{Z}, \mathbf{Z}, NP^+ \rangle$

$\lambda z.|\mathrm{John}|(|\mathrm{loves}|(|\mathrm{Mary}|\, z))$

$\langle \mathbf{X}, \mathbf{X}, S^+ \rangle$

$\langle J, \lambda z.|\mathrm{John}|\, z, NP^- \rangle$

$|\mathrm{loves}|$
$|\mathrm{Mary}|$

Example

- $\langle J, \lambda z.|\mathrm{John}|\, z, NP^- \rangle$
- $\langle M, \lambda z.|\mathrm{Mary}|\, z, NP^- \rangle$
- $\langle L\,\mathbf{Y}\,\mathbf{Z}, \lambda z.\mathbf{Y}(|\mathrm{loves}|(\mathbf{Z}\,z)), S^- \rangle$, $\langle \mathbf{Y}, \mathbf{Y}, NP^+ \rangle$, $\langle \mathbf{Z}, \mathbf{Z}, NP^+ \rangle$

$\lambda z.|\mathrm{John}|(|\mathrm{loves}|(|\mathrm{Mary}|\, z))$

$\langle \mathbf{X}, \mathbf{X}, S^+ \rangle$

$\langle J, \lambda z.|\mathrm{John}|\, z, NP^- \rangle$

$|\mathrm{loves}|$
$|\mathrm{Mary}|$

Example

- $\langle J, \lambda z.|\mathrm{John}|\, z, NP^{-}\rangle$
- $\langle M, \lambda z.|\mathrm{Mary}|\, z, NP^{-}\rangle$
- $\langle L\,\mathbf{Y}\,\mathbf{Z}, \lambda z.\mathbf{Y}(|\mathrm{loves}|(\mathbf{Z}\, z)), S^{-}\rangle$, $\langle \mathbf{Y}, \mathbf{Y}, NP^{+}\rangle$, $\langle \mathbf{Z}, \mathbf{Z}, NP^{+}\rangle$

$$\lambda z.|\mathrm{John}|(|\mathrm{loves}|(|\mathrm{Mary}|\, z))$$

$$\langle \mathbf{X}, \mathbf{X}, S^{+}\rangle$$
$$\langle J, \lambda z.|\mathrm{John}|\, z, NP^{-}\rangle$$
$$\langle L\,\mathbf{Y}\,\mathbf{Z}, \lambda z.\mathbf{Y}(|\mathrm{loves}|(\mathbf{Z}\, z)), S^{-}\rangle$$
$$\langle \mathbf{Y}, \mathbf{Y}, NP^{+}\rangle$$
$$\langle \mathbf{Z}, \mathbf{Z}, NP^{+}\rangle$$

$\lvert\mathrm{Mary}\rvert$

Example

- $\langle J, \lambda z.|\text{John}|\, z, NP^- \rangle$
- $\langle M, \lambda z.|\text{Mary}|\, z, NP^- \rangle$
- $\langle L\,\mathbf{Y}\,\mathbf{Z}, \lambda z.\mathbf{Y}(|\text{loves}|(\mathbf{Z}\,z)), S^- \rangle$, $\langle \mathbf{Y}, \mathbf{Y}, NP^+ \rangle$, $\langle \mathbf{Z}, \mathbf{Z}, NP^+ \rangle$

$\lambda z.|\text{John}|(|\text{loves}|(|\text{Mary}|\, z))$

$\langle \mathbf{X}, \mathbf{X}, S^+ \rangle$

$\langle J, \lambda z.|\text{John}|\, z, NP^- \rangle$

$\langle L\,\mathbf{Y}\,\mathbf{Z}, \lambda z.\mathbf{Y}(|\text{loves}|(\mathbf{Z}\,z)), S^- \rangle$

$\langle \mathbf{Y}, \mathbf{Y}, NP^+ \rangle$

$\langle \mathbf{Z}, \mathbf{Z}, NP^+ \rangle$

$|\text{Mary}|$

Example

- $\langle J, \lambda z.|\mathrm{John}|\, z, NP^{-}\rangle$
- $\langle M, \lambda z.|\mathrm{Mary}|\, z, NP^{-}\rangle$
- $\langle L\,\mathbf{Y}\,\mathbf{Z}, \lambda z.\mathbf{Y}(|\mathrm{loves}|(\mathbf{Z}\,z)), S^{-}\rangle$, $\langle \mathbf{Y}, \mathbf{Y}, NP^{+}\rangle$, $\langle \mathbf{Z}, \mathbf{Z}, NP^{+}\rangle$

$\lambda z.|\mathrm{John}|(|\mathrm{loves}|(|\mathrm{Mary}|\, z))$

$\langle \mathbf{X}, \mathbf{X}, S^{+}\rangle$

$\langle J, \lambda z.|\mathrm{John}|\, z, NP^{\circ}\rangle$

$\langle L\,J\,\mathbf{Z}, \lambda z.|\mathrm{John}|(|\mathrm{loves}|(\mathbf{Z}\,z)), S^{-}\rangle$

$\langle \mathbf{Z}, \mathbf{Z}, NP^{+}\rangle$

$|\mathrm{Mary}|$

Example

- $\langle J, \lambda z.|\text{John}|\, z, NP^{-}\rangle$
- $\langle M, \lambda z.|\text{Mary}|\, z, NP^{-}\rangle$
- $\langle L\,\mathbf{Y}\,\mathbf{Z}, \lambda z.\mathbf{Y}(|\text{loves}|(\mathbf{Z}\, z)), S^{-}\rangle$, $\langle \mathbf{Y}, \mathbf{Y}, NP^{+}\rangle$, $\langle \mathbf{Z}, \mathbf{Z}, NP^{+}\rangle$

$\lambda z.|\text{John}|(|\text{loves}|(|\text{Mary}|\, z))$

$\langle \mathbf{X}, \mathbf{X}, S^{+}\rangle$

$\langle J, \lambda z.|\text{John}|\, z, NP^{\circ}\rangle$

$\langle L\, J\, \mathbf{Z}, \lambda z.|\text{John}|(|\text{loves}|(\mathbf{Z}\, z)), S^{-}\rangle$

$\langle \mathbf{Z}, \mathbf{Z}, NP^{+}\rangle$

$|\text{Mary}|$

Example

- $\langle J, \lambda z.|\text{John}|\, z, NP^- \rangle$
- $\langle M, \lambda z.|\text{Mary}|\, z, NP^- \rangle$
- $\langle L\, \mathbf{Y}\, \mathbf{Z}, \lambda z.\mathbf{Y}(|\text{loves}|(\mathbf{Z}\, z)), S^- \rangle$, $\langle \mathbf{Y}, \mathbf{Y}, NP^+ \rangle$, $\langle \mathbf{Z}, \mathbf{Z}, NP^+ \rangle$

$\lambda z.|\text{John}|(|\text{loves}|(|\text{Mary}|\, z))$

$\langle J, \lambda z.|\text{John}|\, z, NP^\circ \rangle$ $\langle L\, J\, \mathbf{Z}, \lambda z.|\text{John}|(|\text{loves}|(\mathbf{Z}\, z)), S^\circ \rangle$ $\langle \mathbf{Z}, \mathbf{Z}, NP^+ \rangle$	$|\text{Mary}|$

Example

- $\langle J, \lambda z.|\text{John}|\, z, NP^{-}\rangle$
- $\langle M, \lambda z.|\text{Mary}|\, z, NP^{-}\rangle$
- $\langle L\,\mathbf{Y}\,\mathbf{Z}, \lambda z.\mathbf{Y}(|\text{loves}|(\mathbf{Z}\,z)), S^{-}\rangle$, $\langle \mathbf{Y}, \mathbf{Y}, NP^{+}\rangle$, $\langle \mathbf{Z}, \mathbf{Z}, NP^{+}\rangle$

$\lambda z.|\text{John}|(|\text{loves}|(|\text{Mary}|\, z))$

$\langle J, \lambda z.|\text{John}|\, z, NP^{\circ}\rangle$	$|\text{Mary}|$
$\langle L\, J\, \mathbf{Z}, \lambda z.|\text{John}|(|\text{loves}|(\mathbf{Z}\, z)), S^{\circ}\rangle$	
$\langle \mathbf{Z}, \mathbf{Z}, NP^{+}\rangle$	

Example

- $\langle J, \lambda z.|\text{John}|\, z, NP^{-}\rangle$
- $\langle M, \lambda z.|\text{Mary}|\, z, NP^{-}\rangle$
- $\langle L\,\mathbf{Y}\,\mathbf{Z}, \lambda z.\mathbf{Y}(|\text{loves}|(\mathbf{Z}\,z)), S^{-}\rangle$, $\langle \mathbf{Y}, \mathbf{Y}, NP^{+}\rangle$, $\langle \mathbf{Z}, \mathbf{Z}, NP^{+}\rangle$

$\lambda z.|\text{John}|(|\text{loves}|(|\text{Mary}|\, z))$

$\langle J, \lambda z.|\text{John}|\, z, NP^{\circ}\rangle$

$\langle L\, J\, \mathbf{Z}, \lambda z.|\text{John}|(|\text{loves}|(\mathbf{Z}\,z)), S^{\circ}\rangle$

$\langle \mathbf{Z}, \mathbf{Z}, NP^{+}\rangle$

$\langle M, \lambda z.|\text{Mary}|\, z, NP^{-}\rangle$

Example

- $\langle J, \lambda z.|\text{John}|\, z, NP^{-}\rangle$
- $\langle M, \lambda z.|\text{Mary}|\, z, NP^{-}\rangle$
- $\langle L\, \mathbf{Y}\, \mathbf{Z}, \lambda z.\mathbf{Y}(|\text{loves}|(\mathbf{Z}\, z)), S^{-}\rangle$, $\langle \mathbf{Y}, \mathbf{Y}, NP^{+}\rangle$, $\langle \mathbf{Z}, \mathbf{Z}, NP^{+}\rangle$

$\lambda z.|\text{John}|(|\text{loves}|(|\text{Mary}|\, z))$

$\langle J, \lambda z.|\text{John}|\, z, NP^{\circ}\rangle$

$\langle L\, J\, \mathbf{Z}, \lambda z.|\text{John}|(|\text{loves}|(\mathbf{Z}\, z)), S^{\circ}\rangle$

$\langle \mathbf{Z}, \mathbf{Z}, NP^{+}\rangle$

$\langle M, \lambda z.|\text{Mary}|\, z, NP^{-}\rangle$

Example

- $\langle J, \lambda z.|\mathrm{John}|\, z, NP^{-}\rangle$
- $\langle M, \lambda z.|\mathrm{Mary}|\, z, NP^{-}\rangle$
- $\langle L\,\mathbf{Y}\,\mathbf{Z}, \lambda z.\mathbf{Y}(|\mathrm{loves}|(\mathbf{Z}\,z)), S^{-}\rangle$, $\langle \mathbf{Y}, \mathbf{Y}, NP^{+}\rangle$, $\langle \mathbf{Z}, \mathbf{Z}, NP^{+}\rangle$

$\lambda z.|\mathrm{John}|(|\mathrm{loves}|(|\mathrm{Mary}|\, z))$

$\langle J, \lambda z.|\mathrm{John}|\, z, NP^{\circ}\rangle$

$\langle L\,J\,M, \lambda z.|\mathrm{John}|(|\mathrm{loves}|(|\mathrm{Mary}|\, z)), S^{\circ}\rangle$

$\langle M, \lambda z.|\mathrm{Mary}|\, z, NP^{\circ}\rangle$

Example

- $\langle J, \lambda z.|\mathrm{John}|\, z, NP^{-}\rangle$
- $\langle M, \lambda z.|\mathrm{Mary}|\, z, NP^{-}\rangle$
- $\langle L\,\mathbf{Y}\,\mathbf{Z}, \lambda z.\mathbf{Y}(|\mathrm{loves}|(\mathbf{Z}\, z)), S^{-}\rangle$, $\langle \mathbf{Y}, \mathbf{Y}, NP^{+}\rangle$, $\langle \mathbf{Z}, \mathbf{Z}, NP^{+}\rangle$

$\lambda z.|\mathrm{John}|(|\mathrm{loves}|(|\mathrm{Mary}|\, z))$

$\langle J, \lambda z.|\mathrm{John}|\, z, NP^{\circ}\rangle$

$\langle L\; J\; M, \lambda z.|\mathrm{John}|(|\mathrm{loves}|(|\mathrm{Mary}|\, z)), S^{\circ}\rangle$

$\langle M, \lambda z.|\mathrm{Mary}|\, z, NP^{\circ}\rangle$

Properties of incremental Algorithm

- Advantage : No matching equations to solve

Properties of incremental Algorithm

- Advantage : No matching equations to solve
- Disadvantage : one enumerates all the abstract terms the realization of which contains the same constants as u

Properties of incremental Algorithm

- Advantage : No matching equations to solve
- Disadvantage : one enumerates all the abstract terms the realization of which contains the same constants as u
- We have to take into account the structure of u:

Properties of incremental Algorithm

- Advantage : No matching equations to solve
- Disadvantage : one enumerates all the abstract terms the realization of which contains the same constants as u
- We have to take into account the structure of u:
  If $(\mathcal{I}, L, u) \xrightarrow{*}_{Inc} (\mathcal{J}, [], u)$ then $\langle t, \mathcal{L}(t), A^- \rangle \in \mathcal{I}$, $FV(t) = \{x_1; \ldots; x_n\}$ implies the equation

  $$\mathbf{X}\lambda x_1 \ldots x_n.\mathcal{L}(t) \stackrel{?}{=} u$$

  has a solution.

Properties of incremental Algorithm

- Advantage : No matching equations to solve
- Disadvantage : one enumerates all the abstract terms the realization of which contains the same constants as u
- We have to take into account the structure of u:
  If $(\mathcal{I}, L, u) \xrightarrow{*}_{Inc} (\mathcal{J}, [], u)$ then $\langle t, \mathcal{L}(t), A^{-}\rangle \in \mathcal{I}$, $FV(t) = \{x_1; \dots; x_n\}$ implies the equation

  $$\mathbf{X}\lambda x_1 \dots x_n.\mathcal{L}(t) \stackrel{?}{=} u$$

  has a solution.
- In order not to solve matching equations we use a property which is verified whenever $\mathbf{X}\lambda x_1 \dots x_n.\mathcal{L}(t) \stackrel{?}{=} u$ has a solution.

Acceptability and incremental algorithm

As remarked by Morrill [Morrill-2000] such a technique allows us to express the complexity of the analysis of a sentence.

Acceptability and incremental algorithm

As remarked by Morrill [Morrill-2000] such a technique allows us to express the complexity of the analysis of a sentence.
If $(\mathcal{I}_1, L_u, u) \rightarrow_{Inc} \ldots \rightarrow_{Inc} (\mathcal{I}_n, L_u, u)$ is a derivation its complexity is given by:

$$\max_{i \in [1,n]} (|\mathcal{I}_i|^-)$$

where $|\mathcal{I}|^-$ is the maximum number of negative items.

Acceptability and incremental algorithm

As remarked by Morrill [Morrill-2000] such a technique allows us to express the complexity of the analysis of a sentence.
If $(\mathcal{I}_1, L_u, u) \rightarrow_{Inc} \ldots \rightarrow_{Inc} (\mathcal{I}_n, L_u, u)$ is a derivation its complexity is given by:

$$\max_{i \in [1,n]} (|\mathcal{I}_i|^-)$$

where $|\mathcal{I}|^-$ is the maximum number of negative items.
For a given analysis t of u, the complexity of t is the minimal complexity of the derivations which prove t is an analysis of u.

Acceptability and incremental algorithm

As remarked by Morrill [Morrill-2000] such a technique allows us to express the complexity of the analysis of a sentence.
If $(\mathcal{I}_1, L_u, u) \rightarrow_{Inc} \ldots \rightarrow_{Inc} (\mathcal{I}_n, L_u, u)$ is a derivation its complexity is given by:

$$\max_{i \in [1,n]} (|\mathcal{I}_i|^-)$$

where $|\mathcal{I}|^-$ is the maximum number of negative items.
For a given analysis t of u, the complexity of t is the minimal complexity of the derivations which prove t is an analysis of u.
Complexity increases when parsing such phenomena as *garden path*, *left to right quantifier scope*, *centre embedding*... ([Morrill-2000])

Acceptability and incremental algorithm

In order to improve the performance of the algorithm we bound the complexity of the analysis (*i.e.* we bound the number of negative item the set can contain).

Acceptability and incremental algorithm

In order to improve the performance of the algorithm we bound the complexity of the analysis (*i.e.* we bound the number of negative item the set can contain).

- We loose completness but this loss is linguistically motivated

Acceptability and incremental algorithm

In order to improve the performance of the algorithm we bound the complexity of the analysis (*i.e.* we bound the number of negative item the set can contain).

- We loose completness but this loss is linguistically motivated
- we gain polynomiality for the membership problem

Parsing in $\mathcal{L}(2, m)$

- Universal membership problem is NP-complete for grammars of $\mathcal{L}(2, p)$ whenever $p \geq 2$.

Parsing in $\mathcal{L}(2, m)$

- Universal membership problem is NP-complete for grammars of $\mathcal{L}(2, p)$ whenever $p \geq 2$.
- Membership problem is polynomial in $\mathcal{L}(2, m)$.

Parsing in $\mathcal{L}(2, m)$

- Universal membership problem is NP-complete for grammars of $\mathcal{L}(2, p)$ whenever $p \geq 2$.
- Membership problem is polynomial in $\mathcal{L}(2, m)$.
- We show here ideas which enables the construction of an efficient paring algorithm of grammars of $\mathcal{L}(2, m)$.

Parsing in $\mathcal{L}(2, m)$

- Universal membership problem is NP-complete for grammars of $\mathcal{L}(2, p)$ whenever $p \geq 2$.
- Membership problem is polynomial in $\mathcal{L}(2, m)$.
- We show here ideas which enables the construction of an efficient paring algorithm of grammars of $\mathcal{L}(2, m)$.
- This algorithm parses CFGs the same way as Earley algorithm and TAG in $\mathcal{O}(n^6)$.

Parsing in $\mathcal{L}(2, m)$

- Universal membership problem is NP-complete for grammars of $\mathcal{L}(2, p)$ whenever $p \geq 2$.
- Membership problem is polynomial in $\mathcal{L}(2, m)$.
- We show here ideas which enables the construction of an efficient paring algorithm of grammars of $\mathcal{L}(2, m)$.
- This algorithm parses CFGs the same way as Earley algorithm and TAG in $\mathcal{O}(n^6)$.
- In order to remain simple we restrict ouselves to the case where the object signature is of the second order (object terms are trees).

Syntactic descriptions

Syntactic descriptions are the mathematic abstraction guiding the algorithm:

$$\mathcal{D} ::= \{\mathcal{T} : T_{\multimap}\} \mid \mathcal{D} \multimap \mathcal{D}$$

$[\![d]\!]$ is the semantics of d:

- $[\![\{t : \alpha\}]\!] = \{v \mid \cdot \vdash v : \alpha \wedge v =_{\beta\eta} t\}$
- $[\![d_1 \multimap d_2]\!] = \{v \mid \forall w \in [\![d_1]\!].(vw) \in [\![d_2]\!]\}$

a description d is complete if:

- $d = \{t : \alpha\}$ and t is in β-normal η-long form and α is atomic
- $d = d_1 \multimap d_2$ and d_1 and d_2 are complete

Syntactic descriptions: examples

Syntactic descriptions enables to model higher-order contexts:

Given a string $a_1(\ldots(a_n(e))\ldots)$, the description:

$$\{a_j(\ldots(a_n e)\ldots) : *\} \multimap \{a_i(\ldots(a_n e)\ldots) : *\}$$

represents the substring $\lambda x.a_i(\ldots(a_j x)\ldots)$

Syntactic descriptions: examples

Syntactic descriptions enables to model higher-order contexts:

Given a string $a_1(\ldots(a_n(e))\ldots)$, the description:

$$\{a_j(\ldots(a_n e)\ldots):*\} \multimap \{a_i(\ldots(a_n e)\ldots):*\}$$

represents the substring $\lambda x.a_i(\ldots(a_j x)\ldots)$

The description:

$$(|a_k \ldots a_n| \multimap |a_j \ldots a_n|) \multimap (|a_l \ldots a_n| \multimap |a_i \ldots a_n|)$$

$$|a_p \ldots a_n| = \{a_p(\ldots(a_n e)\ldots):*\}$$

Syntactic descriptions: examples

Syntactic descriptions enables to model higher-order contexts:
Given a string $a_1(\ldots(a_n(e))\ldots)$, the description:

$$\{a_j(\ldots(a_n e)\ldots) : *\} \multimap \{a_i(\ldots(a_n e)\ldots) : *\}$$

represents the substring $\lambda x.a_i(\ldots(a_j x)\ldots)$
The description:

$$\underbrace{(|a_k \ldots a_n| \multimap |a_j \ldots a_n|)}_{a_j \ldots a_k} \multimap (|a_l \ldots a_n| \multimap |a_i \ldots a_n|)$$

$$|a_p \ldots a_n| = \{a_p(\ldots(a_n e)\ldots) : *\}$$

Syntactic descriptions: examples

Syntactic descriptions enables to model higher-order contexts:

Given a string $a_1(\dots(a_n(e))\dots)$, the description:

$$\{a_j(\dots(a_n e)\dots) : *\} \multimap \{a_i(\dots(a_n e)\dots) : *\}$$

represents the substring $\lambda x.a_i(\dots(a_j x)\dots)$

The description:

$$\underbrace{(|a_k \dots a_n| \multimap |a_j \dots a_n|)}_{a_j \dots a_k} \multimap \underbrace{(|a_l \dots a_n| \multimap |a_i \dots a_n|)}_{a_i \dots a_l}$$

$|a_p \dots a_n| = \{a_p(\dots(a_n e)\dots) : *\}$

Syntactic descriptions: examples

Syntactic descriptions enables to model higher-order contexts:

Given a string $a_1(\ldots(a_n(e))\ldots)$, the description:

$$\{a_j(\ldots(a_n e)\ldots) : *\} \multimap \{a_i(\ldots(a_n e)\ldots) : *\}$$

represents the substring $\lambda x.a_i(\ldots(a_j x)\ldots)$

The description:

$$\underbrace{(|a_k \ldots a_n| \multimap |a_j \ldots a_n|)}_{a_j \ldots a_k} \multimap \underbrace{(|a_l \ldots a_n| \multimap |a_i \ldots a_n|)}_{a_i \ldots a_l}$$

$|a_p \ldots a_n| = \{a_p(\ldots(a_n e)\ldots) : *\}$

represents the context

$$a_1 \ldots \underline{a_i \ldots a_j} \ldots \underline{a_k \ldots a_l} \ldots a_n$$

Syntactic descriptions: examples

Syntactic descriptions enables to model higher-order contexts:
Given a string $a_1(\ldots(a_n(e))\ldots)$, the description:

$$\{a_j(\ldots(a_n e)\ldots):*\} \multimap \{a_i(\ldots(a_n e)\ldots):*\}$$

represents the substring $\lambda x.a_i(\ldots(a_j x)\ldots)$
The description:

$$\underbrace{(|a_k\ldots a_n| \multimap |a_j\ldots a_n|)}_{a_j\ldots a_k} \multimap \underbrace{(|a_l\ldots a_n| \multimap |a_i\ldots a_n|)}_{a_i\ldots a_l}$$

$|a_p\ldots a_n| = \{a_p(\ldots(a_n e)\ldots):*\}$
represents the context

$$a_1\ldots \underline{a_i\ldots a_j}\ldots \underline{a_k\ldots a_l}\ldots a_n$$

Descriptions model the indices involved in Earley parsing

Sequent for parsing

For parsing we use sequent of the form: $\Gamma; \Delta \vdash_D t : d$

Sequent for parsing

For parsing we use sequent of the form: $\Gamma; \Delta \vdash_D t : d$ where:

- Γ is a context which assigns atomic abstract type to variables

Sequent for parsing

For parsing we use sequent of the form: $\Gamma; \Delta \vdash_D t : d$ where:

- Γ is a context which assigns atomic abstract type to variables
- Δ is a context which associates descriptions to variable

Formal system for parsing

$$\frac{\cdot \vdash_L t : \alpha}{\cdot \vdash_A \{t : \alpha\} : \mathrm{Type}} \qquad \frac{\cdot \vdash_A d_1 : \mathrm{Type} \quad \cdot \vdash_A d_2 : \mathrm{Type}}{\cdot \vdash_A d_1 \multimap d_2 : \mathrm{Type}} \qquad \frac{\cdot \vdash_A d : \mathrm{Type}}{\cdot; x : d \vdash_A x : d}$$

$$\frac{\Gamma_1; \Delta_1 \vdash_A t_1 : \{v_1 : \alpha \multimap \beta\} \quad \Gamma_2; \Delta_2 \vdash_A t_2 : \{v_2 : \alpha\}}{\Gamma_1, \Gamma_2; \Delta_1, \Delta_2 \vdash_A t_1 t_2 \{v_1 v_2 : \beta\}}$$

$$\frac{x_1 : A_1, \ldots, x_n : A_n; \cdot \vdash_A t : d \quad \lambda x_1 \ldots x_n.t \in \mathcal{L}^{A_1 \multimap \ldots \multimap A_n \multimap A}}{x : A; \cdot \vdash x : d}$$

$$\frac{\Gamma; \Delta, x : d_1 \vdash_A t : d_2}{\Gamma; \Delta \vdash_A \lambda x.t : d_1 \multimap d_2} \qquad \frac{\Gamma_1; \Delta_1 \vdash_A t_1 : d_1 \multimap d_2 \quad \Gamma_2; \Delta_2 \vdash_A t_2 : d_1}{\Gamma_1, \Gamma_2; \Delta_1, \Delta_2 \vdash_A t_1 t_2 : d_2}$$

Property of the system

If t has an atomic type and is in β-normal η-long form and all the descriptions are complete, then the sequent

$$x_1 : A_1, \ldots, x_n : A_n; y_1 : d_1, \ldots, y_n : d_n \vdash_A t : \{u : *\}$$

is derivable iff:
there are abstract terms $(t_i)_{i \in [1,n]}$ such that t_i has type A_i
for all terms $(v_i)_{i \in [1,n]}$ such that $v_i \in [\![d_i]\!]$,

$$t[x_1 := \mathcal{L}(t_1); \ldots; x_n := \mathcal{L}(t_n); y_1 := v_1; \ldots y_n := v_n] \in [\![\{u : *\}]\!]$$

Parsing principle

To obtain the algorithm we use:

- a chart of items
- the items which represent sequents $\Gamma; \Delta \vdash_A t : \{v : o\}$ where t is the subterm of a lexical entry
- rules which emulate the formal system

N.B: this algorithm can easily be extended all the grammars of $\mathcal{L}(2, m)$

Conclusion

- there is a semi-algorithm which can parse any ACG. But it is not efficient

Conclusion

- there is a semi-algorithm which can parse any ACG. But it is not efficient
- incremental parsing suits well to natural language

Conclusion

- there is a semi-algorithm which can parse any ACG. But it is not efficient
- incremental parsing suits well to natural language
- efficient parsing of grammars of $\mathcal{L}(2, m)$ is possible

Conclusion

- there is a semi-algorithm which can parse any ACG. But it is not efficient
- incremental parsing suits well to natural language
- efficient parsing of grammars of $\mathcal{L}(2, m)$ is possible
- try to shift the technology of description for parsing any ACG (proving and matching collaborate to parsing)

Bibliography

References

[Morrill-2000] Glyn Morrill, Incremental Processing and Acceptability, *Computational Linguistics*, 2000, 26, 3, 319-338.

[de Groote-2000] Philippe de Groote Proof-Search in Implicative Linear Logic as a Matching Problem. *LPAR*, 2000, 257-274.

[de Groote-2001] Philippe de Groote, Towards Abstract Categorial Grammars, *ACL*, 2001, 148-155.