On Learning Discontinuous Dependencies from Positive Data

Denis Béchet[(1)], Alexander Dikovsky[(2)], Annie Foret[(3)] and Erwan Moreau[(2)]

[(1)] LIPN, University Paris 13

[(2)] LINA, University of Nantes

[(3)] IRISA, University of Rennes

Gold's Model : identification in the limit

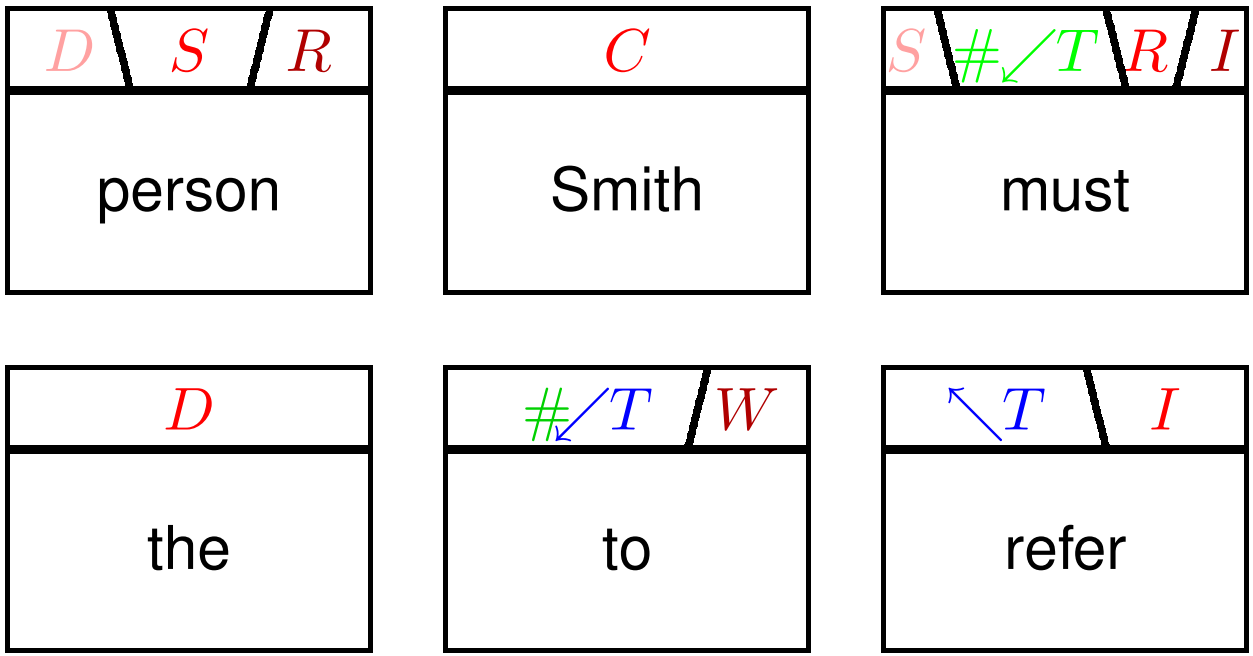
- Algorithm:
  Input : a finite set of sentences (positive examples)
  Output : a grammar in the class that generates the sentences ; *the algorithm is required to converge*
- Formally:
  - $\mathcal{G}$: class of grammars
  - Σ : alphabet
  - ϕ : function from finite subsets of Σ^* to $\mathcal{G}$ such that
    $\forall G \in \mathcal{G}$, $\forall \langle e_i \rangle_{i \in \mathbb{N}}$ with $\mathcal{L}(G) = \langle e_i \rangle_{i \in \mathbb{N}}$:
    $\exists G' \in \mathcal{G}$ with $\mathcal{L}(G') = \mathcal{L}(G)$
    $\exists n_0 \in N$: $\forall n > n_0$ $\phi(\{e_1, \ldots, e_n\}) = G' \in \mathcal{G}$
    where $\mathcal{L}(G)$ *denotes the language associated to* G

PLAN

- Introduction
- Background [Dikovsky 2004]
  - Categories, dependencies, trees and nets
  - Categorial dependency grammars (CDGs)
  - Languages of (untyped) nets/depencency trees/strings
- Learning k-valued/rigid CDGs
  - Learning from untyped nets: an algorithm
  - Learning from untyped nets: finite elasticity
  - Learning from strings: finite elasticity (CDGs without optional or iterative types)
  - Non-Learnability for CDGs with optional or iterative types
- Conclusion

Categories, nodes, dependencies and nets

Categories and nodes:



$D = det$

$S = subj$

$R = attr\text{–}rel$

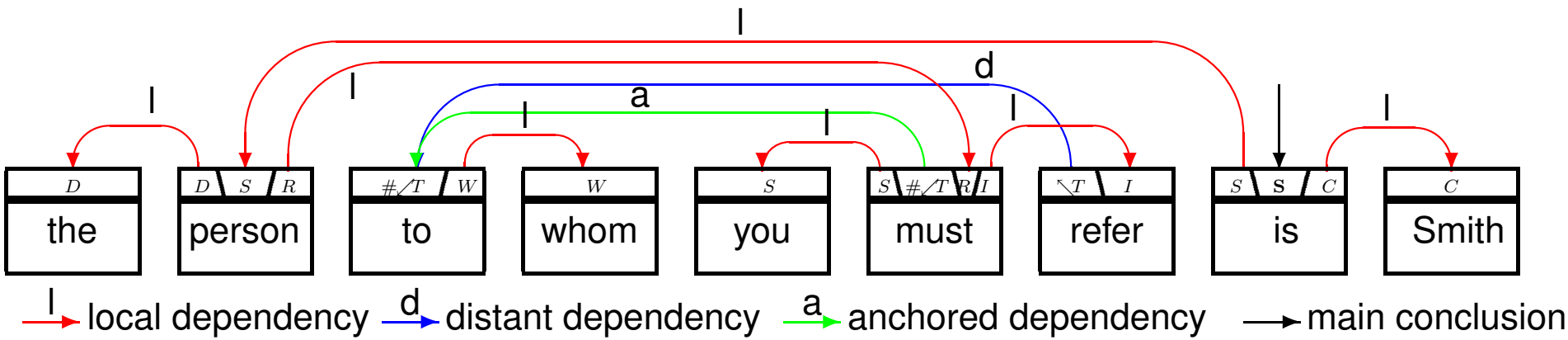
$C = n\text{–}copul$

$T = pre\text{–}TO\text{–}obj$

$W = prep\text{–}wh$

$I = inf\text{–}obj$

An untyped net:



CDG: definitions (1)

- Σ = alphabet for words in a natural language
- $\mathbf{C}$ = set of elementary categories : $det, subj, \cdots$
- $Start(\mathbf{C})$ and $End(\mathbf{C})$ = starting and ending atoms :
  - C : start or end of 1 local dependency
  - C^* : start of 0, 1 or more local dependencies (iterative)
  - C^+ : start of 1 or more local dependencies (repetitive)
  - $C^?$: start 0 or 1 local dependency (optional)
  - $\nwarrow C$: start (on the left) of 1 distant dependency
  - $\swarrow C$: end (from the right) of 1 distant dependency
  - $\#\swarrow C$: start of 1 anchored dependency or end of 1 anchored dependency and of 1 distant dependency
  - $\nearrow C$, $\searrow C$ and $\#\searrow C$ for the reverse direction

CDG: definitions (2)

- $Cat(\mathbf{C})$ = categories = $List(Start(\mathbf{C})) \times End(\mathbf{C}) \times List(Start(\mathbf{C}))$ written $L_1 \backslash \cdots \backslash L_i \backslash C / R_j / \cdots / R_1$
- $\vdash$ = provability relation based on simplification rules (local, distant and anchored dependencies):

$$
\begin{array}{lcll}
\Gamma_1 C[C\backslash\alpha]\Gamma_2 & \vdash & \Gamma_1\alpha\Gamma_2 & (\mathbf{L}) \\
\Gamma_1 C[C^*\backslash\alpha]\Gamma_2 & \vdash & \Gamma_1[C^*\backslash\alpha]\Gamma_2 & (\mathbf{I}) \\
\Gamma_1 C[C^+\backslash\alpha]\Gamma_2 & \vdash & \Gamma_1[C^*\backslash\alpha]\Gamma_2 & (\mathbf{R}) \\
\Gamma_1 C[C^?\backslash\alpha]\Gamma_2 & \vdash & \Gamma_1\alpha\Gamma_2 & (\mathbf{O}) \\
\Gamma_1[C^{?|*}\backslash\alpha]\Gamma_2 & \vdash & \Gamma_1\alpha\Gamma_2 & (\mathbf{\Omega}) \\
\Gamma_1\#(\alpha)[\#(\alpha)\backslash\beta]\Gamma_2 & \vdash & \Gamma_1\alpha\beta\Gamma_2 & (\mathbf{A}) \\
\Gamma_1(\swarrow C)\Gamma_2^{(1)}[(\nwarrow C)\backslash\alpha]\Gamma_3 & \vdash & \Gamma_1\Gamma_2\alpha\Gamma_3 & (\mathbf{D})
\end{array}
$$

(1) Γ_2 without $\swarrow C, \nwarrow C$ and $\#\swarrow C$

CDG: definitions (3)

- A *CDG* is a finite relation G between Σ and $Cat(\mathbf{C})$
- G *generates* a string $c_1 \ldots c_n \in \Sigma^+$ iff
  $\exists A_1, \ldots, A_n \in Cat(C)$ such that:
  $G : c_i \mapsto A_i \ (1 \leq i \leq n)$ and $A_1 \cdots A_n \vdash S$
- The language of G,
  $\mathcal{L}(G)$ = the set of strings generated by G
- G k-valued iff at most k types per symbol (rigid if $k = 1$)

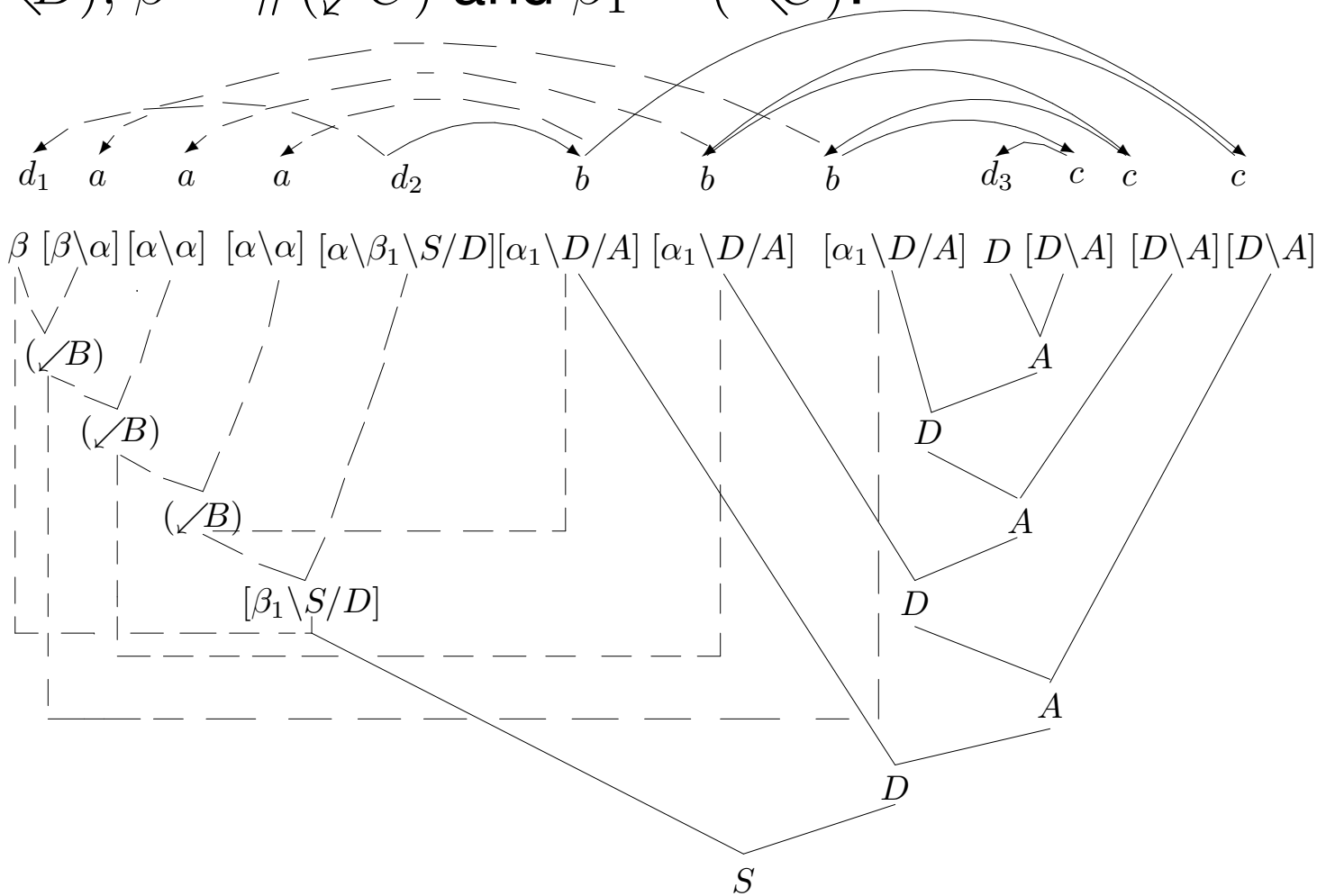
Example of a CDG (1)

$$G_0 : \begin{cases} a & \mapsto & \#\swarrow C \backslash \#\swarrow B \ \mid \ \#\swarrow B \backslash \#\swarrow B \\ b & \mapsto & \nwarrow B \backslash D \\ c & \mapsto & D \backslash A \\ d_1 & \mapsto & \#\swarrow B \\ d_2 & \mapsto & \#\swarrow B \backslash \nwarrow C \backslash S / D \\ d_3 & \mapsto & D \end{cases}$$

$\mathcal{L}(G_0) = \{d_1 a^n d_2 b^n d_3 c^n \mid n > 0\}$: a non-CF language

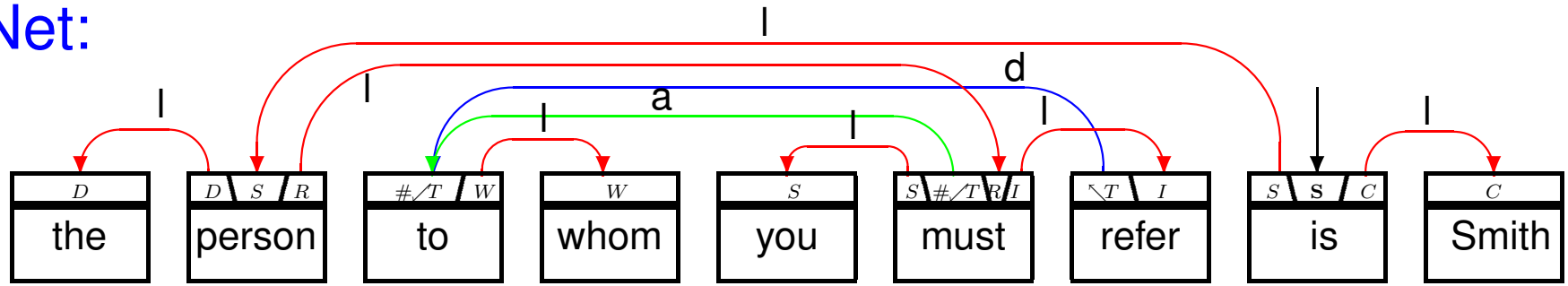
Example of a CDG (2)

A proof of $d_1a^3d_2b^3d_3c^3 \in L(G_0)$, in which $\alpha = \#(\swarrow B)$, $\alpha_1 = (\nwarrow B)$, $\beta = \#(\swarrow C)$ and $\beta_1 = (\nwarrow C)$:

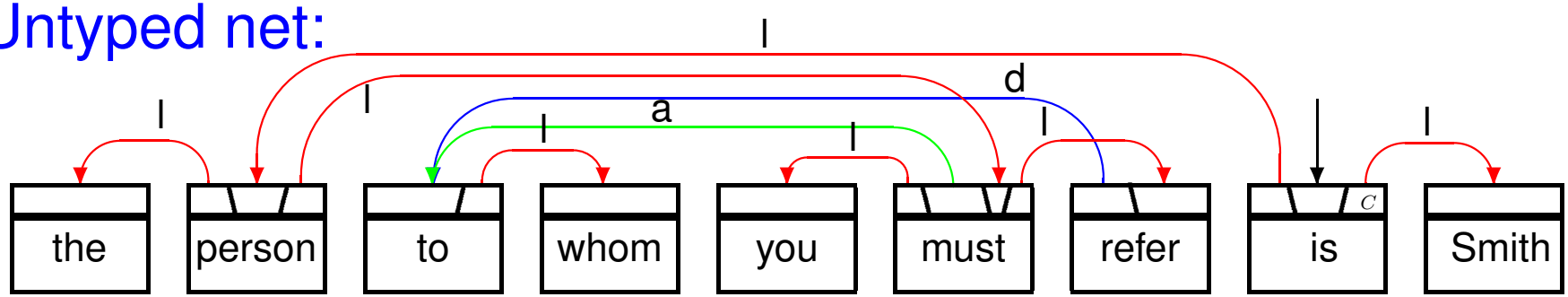


Net, untyped net, tree or string

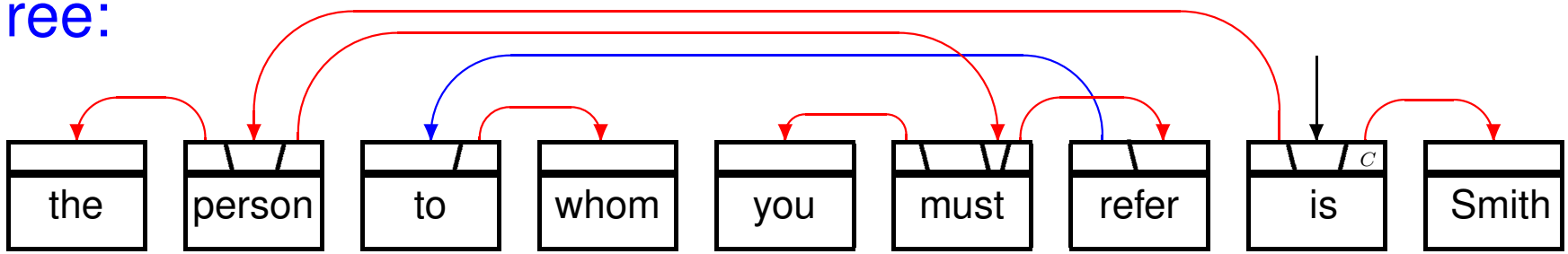
Net:



Untyped net:



Tree:



String:

the person to whom you must refer is Smith

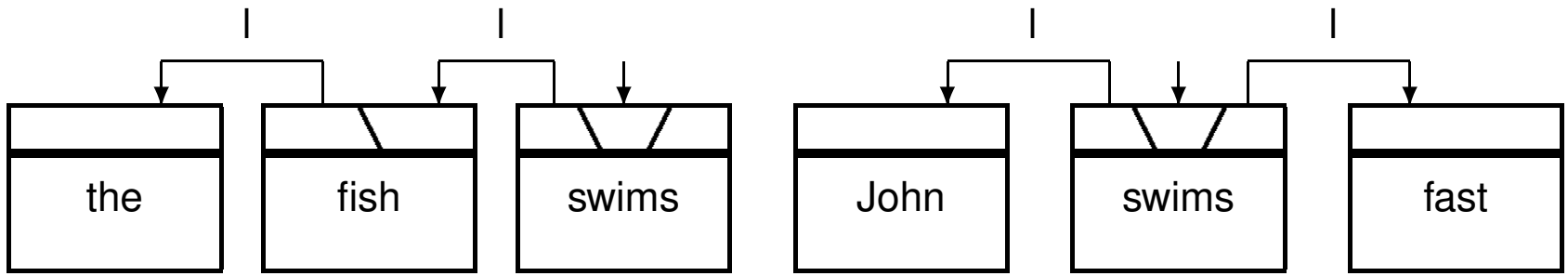
PLAN

- Introduction
- Background [Dikovsky 2004]
  - Categories, dependencies, trees and nets
  - Categorial dependency grammars (CDGs)
  - Languages of (untyped) nets/depencency trees/strings
- Learning k-valued/rigid CDGs
  - Learning from untyped nets: an algorithm
  - Learning from untyped nets: finite elasticity
  - Learning from strings: finite elasticity (CDGs without optional or iterative types)
  - Non-Learnability for CDGs with optional or iterative types
- Conclusion

Learning from untyped nets

(Rigid grammars)

Starting with the following positive examples

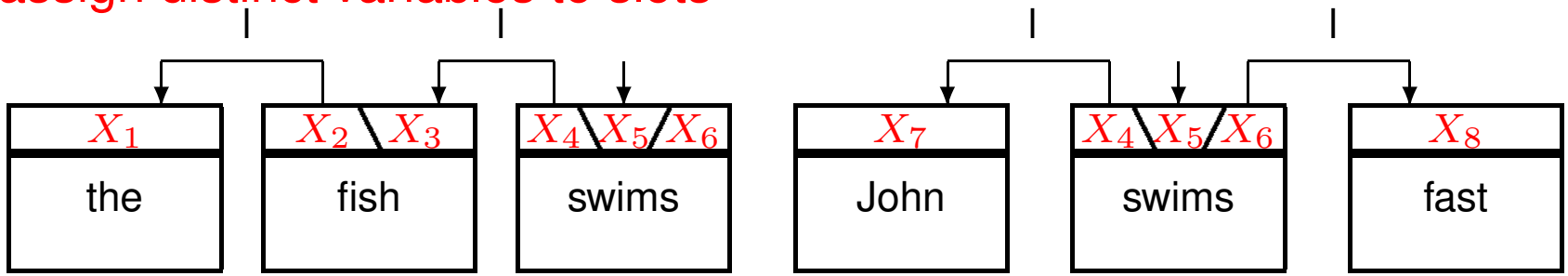


Learning from untyped nets

(Rigid grammars)

Starting with the following positive examples

we assign distinct variables to slots

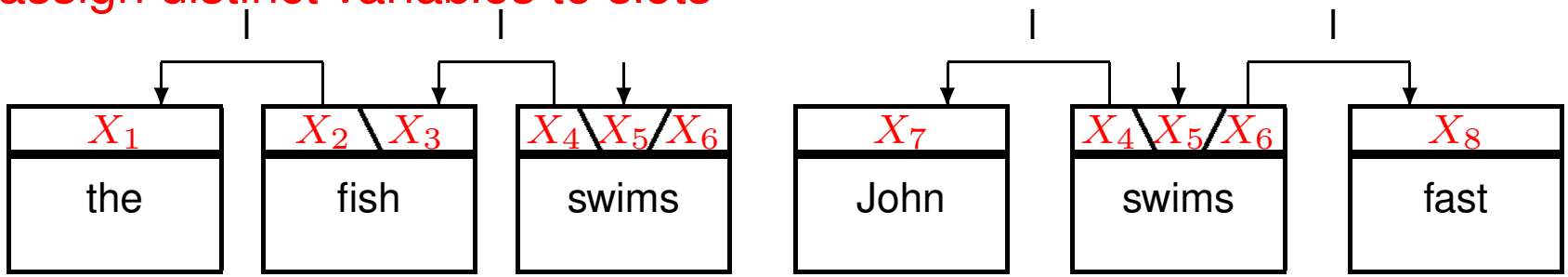


Learning from untyped nets

(Rigid grammars)

Starting with the following positive examples

we assign distinct variables to slots



and obtain the resulting grammar after unification (and link counting)

	General form	Unification	Rigid grammar
the	X_1		X_1
$fish$	$X_2 \backslash X_3$	$X_2 = X_1$	$X_1 \backslash X_3$
$swims$	$X_4 \backslash X_5 / X_6$	$X_4 = X_3; X_5 = S$	$X_3 \backslash S / X_6^?$
$John$	X_7	$X_7 = X_4 (= X_3)$	X_3
$fast$	X_8	$X_8 = X_6$	X_6

Finite elasticity and learnability

Infinite elasticity:

$\exists (e_i)_{i \in \mathbb{N}}$ an infinite sequence of sentences
$\exists (L_i)_{i \in \mathbb{N}}$ an infinite sequence of languages

$$\text{such that } \begin{cases} \forall i \in \mathbb{N} \ : \ e_i \notin L_i \\ \{e_0, \ldots, e_{i-1}\} \subseteq L_i \end{cases}$$

Properties [Wright 1989]

- unlearnability $\Rightarrow$ infinite elasticity
- finite elasticity $\Rightarrow$ learnability

Rigid languages of untyped nets

Theorem:

1. Finite elasticity (in fact *bounded finite thickness*)
2. The class is learnable
3. The given algorithm learns the class

Proof schemes:

(1) There is a finite number of "reduced" languages in the class that correspond to a given list of untyped nets
(2) Consequence of finite elasticity
(3) The returned grammar corresponds to the minimal language compatible with the input untyped nets

Proof of finite elasticity (1)

[Shinohara 1990]: finite thickness $\Rightarrow$ finite elasticity

- **Definition.** $G_1 \sqsubseteq G_2$ iff
  $\forall x \in \Sigma, C \in Cat(\mathbf{C}), G_1 : x \mapsto C \Rightarrow G_2 : x \mapsto C$
- **Definition and lemma.** The mapping $\mathcal{L}_{UNet}$ from link grammars to untyped net languages is monotonic: if $G_1 \sqsubseteq G_2$ then $\mathcal{L}_{UNet}(G_1) \subseteq \mathcal{L}_{UNet}(G_2)$
- **Definition.** A grammar G is reduced with respect to a set X of untyped nets iff $X \subseteq \mathcal{L}_{UNet}(G)$ and for each grammar $G' \sqsubseteq G$, $G' \neq G \Rightarrow X \not\subseteq \mathcal{L}_{UNet}(G')$

Proof of finite elasticity (2)

- **Lemma.** For each finite set $X \subseteq \mathcal{L}_{UNet}(G)$, there is a finite set of rigid untyped net languages that correspond to the grammars that are reduced from X
- **Definition.** Monotonicity and the previous property define a system that has bounded finite thickness
- **Theorem [Shinohara 1990].** A formal system that has bounded finite thickness has finite elasticity
- **Corollary.** Rigid untyped net languages have finite elasticity

k-valued languages of strings (1)

Theorem [Kanasawa 1998]:

The images, through a finite-valued relation of a class of languages that has finite elasticity, forms a class that has finite elasicity

Consequences:

1. For each k, the class of k-valued CDG (without optional and iterative category) languages of strings has finite elasticity
2. For each k, this class (of CDGs) is learnable from strings

k-valued languages of strings (2)

Lemma:

1. For each k, the classes of k-valued CDG without optional and/or iterative category languages of strings accept a limit point
2. For each k, these classes (of CDGs) are not learnable from strings

Conclusion

- Rigid CDGs are learnable from untyped nets
- There exists a simple learning algorithm for this class
- Theoretical learnability result (finite elasticity) for k-valued CDGs without optional and iterative category learned from strings
- Rigid (or k-valued) CDGs with optional and/or iterative category are not learnable from strings

Class	Learnable from strings		Finite elasticity on strings		Finite elasticity on structures		Finite-valued relation
A^*	no	$\Rightarrow$	no		yes	$\Rightarrow$	no
$A^?$	no	$\Rightarrow$	no		yes	$\Rightarrow$	no
A^+	yes	$\Leftarrow$	yes	$\Leftarrow$	yes		yes