# Introduction à l'inférence grammaticale

Isabelle Tellier

LIFO, Université d'Orléans

# Outline

1. Introduction : historical motivations
2. The learnability according to Gold
3. Categorial Grammars and their properties
4. Learning CG by generalization
5. Learning CG by specialization
6. Conclusion

# Introduction : historical motivations

Learnability of natural languages and other things

- in the 1960ies : controversies about natural language acquisition
  - "behaviorists" consider the mind as a black box : learning results from conditionning (stimulus-response)
  - Chomsky argues about the poverty of the (linguistic) stimulus
  - he concludes there exists an innate human capability to acquire formal grammars
- first researchs in the domain of inductive inference : how is it possible to continue a sequence like 0, 2, 4, 6... ? (Solomonoff, Kolmogorov...)

$\Longrightarrow$ need to formalize the notion of learnability

# Outline

1. Introduction : historical motivations
2. The learnability according to Gold
3. Categorial Grammars and their properties
4. Learning CG by generalization
5. Learning CG by specialization
6. Conclusion

# The learnability according to Gold

General requirements for NL learning

- inputs : syntactically correct sentences (and incorrect ones ?) belonging to a language
- target : a formal grammar generating this language
- learnability concerns classes of grammars and not a single one
- a class is learnable if there exists a learning algorithm to identify any of its members
- the learning process is a never-ending one :

$$\begin{array}{cccccc} e_1 & e_2 & e_3 & e_4 & e_5 & \dots \\ \downarrow & \downarrow & \downarrow & \downarrow & \downarrow & \dots \\ G_1 & G_2 & G_3 & G_4 & G_5 & \dots \end{array}$$

# The learnability according to Gold

Learnability "in the limit" from positive examples model (Gold 67)

- $\mathcal{G}$ : a class (set) of grammars
- $L(G)$ denotes the language of strings/structures of $G \in \mathcal{G}$
- the learner algorithm $\phi$ learns $\mathcal{G}$ if :
  - $\forall G \in \mathcal{G}$
  - $\forall \{e_i\}_{i\in\mathbb{N}}$ with $L(G) = \{e_i\}_{i\in\mathbb{N}}$

  $\exists G' \in \mathcal{G}$ with $L(G') = L(G)$

  $\exists n_0 \in \mathbb{N} : \forall n > n_0 \;\; \phi(\{e_1, \ldots, e_n\}) = G' \in \mathcal{G}$
- $\mathcal{G}$ is learnable in the limit if there exists $\phi$ that learns $\mathcal{G}$
- if no such algorithm exists, $\mathcal{G}$ is not learnable

# The learnability according to Gold

First results

– with positive and negative examples : every recursively enumerable class is learnable with a stupid enumeration algorithm

– with positive examples only : if a class generates every finite language plus at least an infinite one, it is not learnable

– example : let $\Sigma = \{a\}$

  – the set of every finite language on $\Sigma$ is $\mathcal{L}$

  – the target class is $\mathcal{L} \cup \{a^*\}$

  – let a sequence of examples : $aaa$, $a$, $aaaaaaaaaaa$, $a$, $aa$,...

  – if the algorithm chooses the generator of a finite language, it will never find $a^*$

  – if the algorithm chooses $a^*$, it may overgeneralize but will never receive a counterexample

$\Longrightarrow \mathcal{L} \cup \{a^*\}$ is not learnable from positive examples

# The learnability according to Gold

Problems and heritage of Gold's definition

- a class can be proved learnable without explicitely providing a learning algorithm (a default enumerating one is enough)
- no complexity criterion is required for the learning process
- direct consequence of the first result : none of the class in the Chomsky hierarchy is learnable from positive examples
- neglected for a time, Gold's definition revived in the 80ies
- interesting new results include :
  - the definition of learnable classes of grammars transversal to the Chomsky hierarchy : Angluin 80, Kanazawa 98
  - the definition of original learning algorithms

# The learnability according to Gold

The main two possible strategies

- available data : a set of positive examples, the target class
- learning by generalization :
  - build a least general grammar generating the examples
  - apply a generalization operator until it belongs to the target class
- learning by specialization :
  - the initial hypothesis space is the whole target class
  - use the examples to constrain this space until it is reduced to one grammar

# Outline

1. Introduction : historical motivations
2. The learnability according to Gold
3. Categorial Grammars and their properties
4. Learning CG by generalization
5. Learning CG by specialization
6. Conclusion

# Categorial Grammars and their properties

Definition of a AB-Categorial Grammar

- a finite vocabulary : $\Sigma = \{$John, runs, fast, a, man$\}$
- a set of basic categories among which is the axiom $S$ :
  $\mathcal{B} = \{S, T, CN\}$ ($T$ for "term", $CN$ for "common noun")
- the set of available categories is the set of oriented fraction over categories : $T\backslash S$, $(S/(T\backslash S))/CN$...
- a Categorial Grammar is set of associations (word,category) :

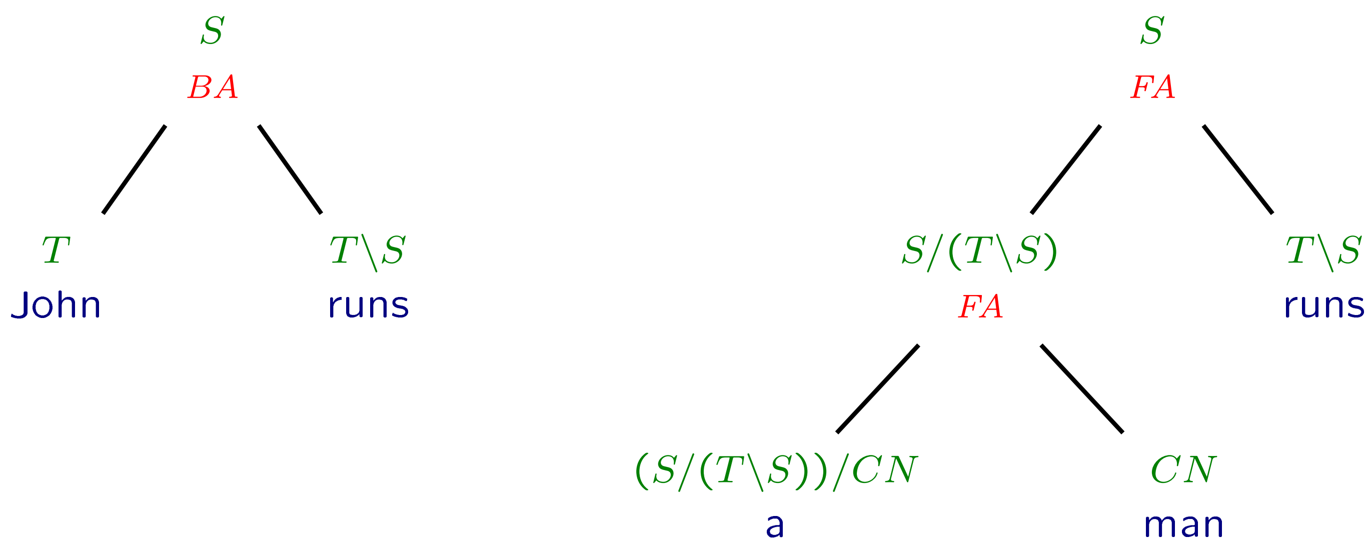
| word | category |
|---|---|
| John | $T$ |
| runs | $T\backslash S$ |
| fast | $(T\backslash S)\backslash(T\backslash S)$ |
| man | $CN$ |
| a | $(S/(T\backslash S))/CN$ |

# Categorial Grammars and their properties

## Language of a AB-Categorial Grammar

- Syntactic rules are expressed by two schemes : $\forall A, B \in Cat(\mathcal{B})$
  - Forward Application $FA$ : $A/B \; B \longrightarrow A$
  - Backward Application $BA$ : $B \; B\backslash A \longrightarrow A$
- a string of words is syntactically correct if a corresponding sequence of categories reduces to $S$

## Example



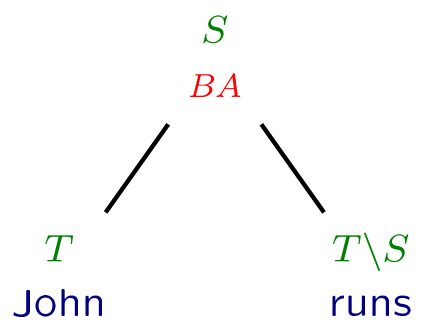
# Categorial Grammars and their properties

AB-Categorial Grammars are well adapted to natural languages (Oehrle, Bach & Wheeler 88) because :

- they are lexicalized
- they have a good expressivity : $\epsilon$-free context-free languages (Bar-Hillel, Gaifman, Shamir 60)
- they can be compositionally linked with formal semantics (Montague 74, Moortgat 88) :
  - a morphism $h$ transforms each syntactic category into a semantic type
  - a translation function associates to each couple (word, category) a logical formula of the right type
  - each syntactic scheme of rule is transformed into a semantic composition rule

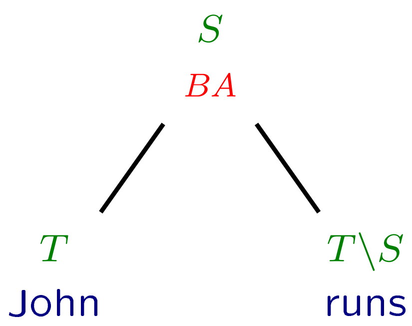
# Categorial Grammars and their properties

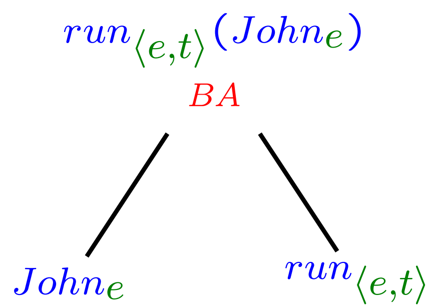
- elementary types : $t$ (type of truth values) and $e$ (type of entities)
- $h(S) = t$, $h(T) = e$, $h(CN) = \langle e, t \rangle$ (one-place predicate)
- for any category $A, B$ : $h(A/B) = h(B\backslash A) = \langle h(B), h(A) \rangle$
- each couple (word, categorie) is translated into a logical formula



# Categorial Grammars and their properties

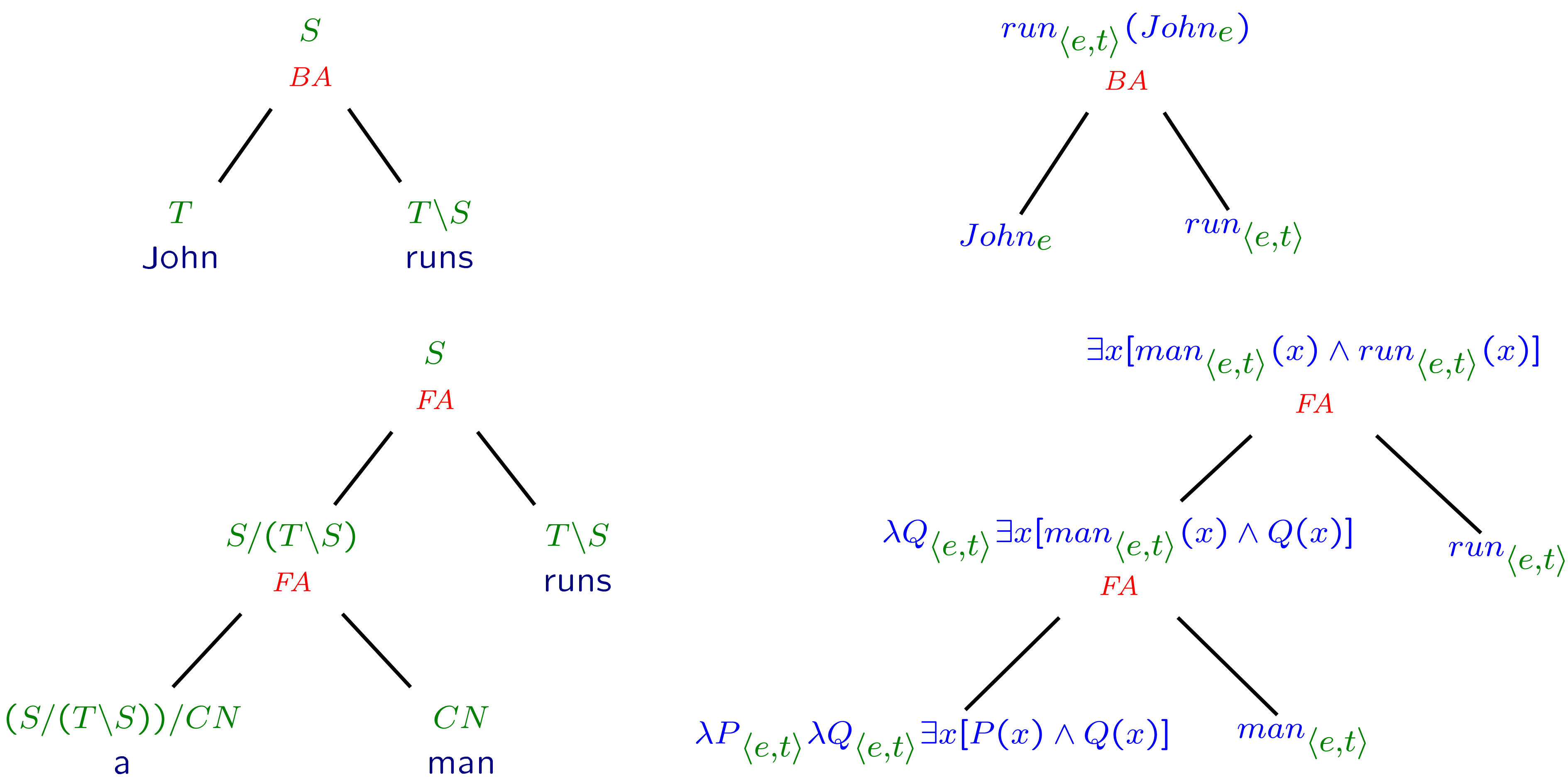
- elementary types : $t$ (type of truth values) and $e$ (type of entities)
- $h(S) = t$, $h(T) = e$, $h(CN) = \langle e, t\rangle$ (one-place predicate)
- for any category $A, B$ : $h(A/B) = h(B\backslash A) = \langle h(B), h(A)\rangle$
- each couple (word, categorie) is translated into a logical formula



# Categorial Grammars and their properties

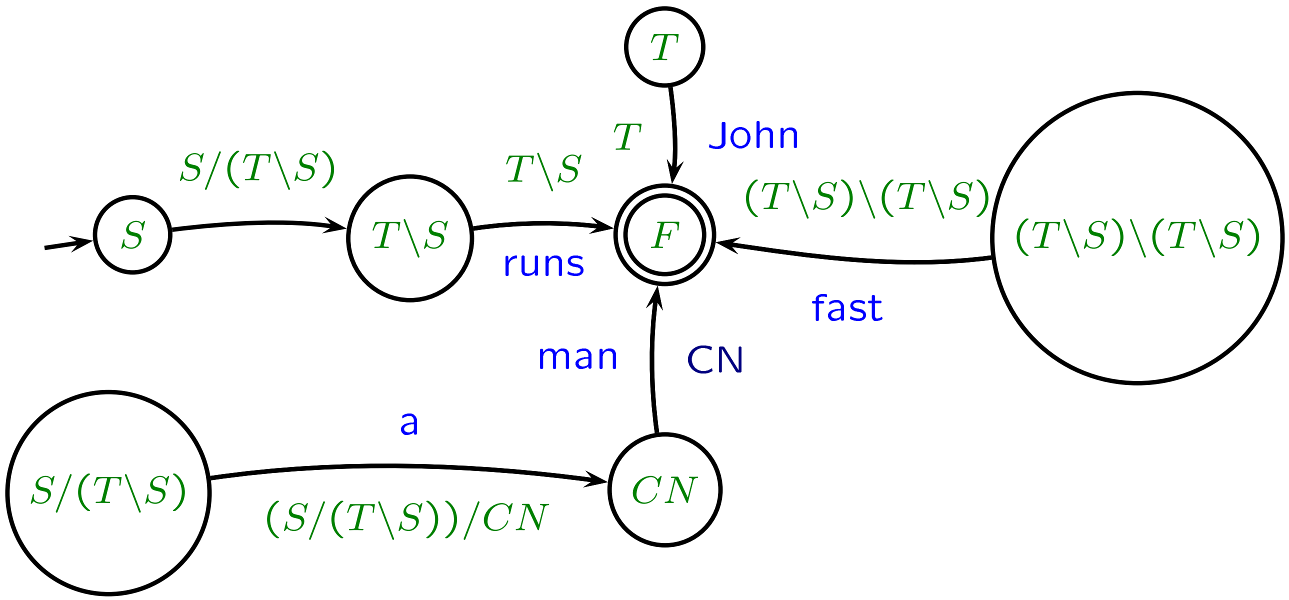
- elementary types : $t$ (type of truth values) and $e$ (type of entities)
- $h(S) = t$, $h(T) = e$, $h(CN) = \langle e, t \rangle$ (one-place predicate)
- for any category $A, B$ : $h(A/B) = h(B\backslash A) = \langle h(B), h(A) \rangle$
- each couple (word, categorie) is translated into a logical formula



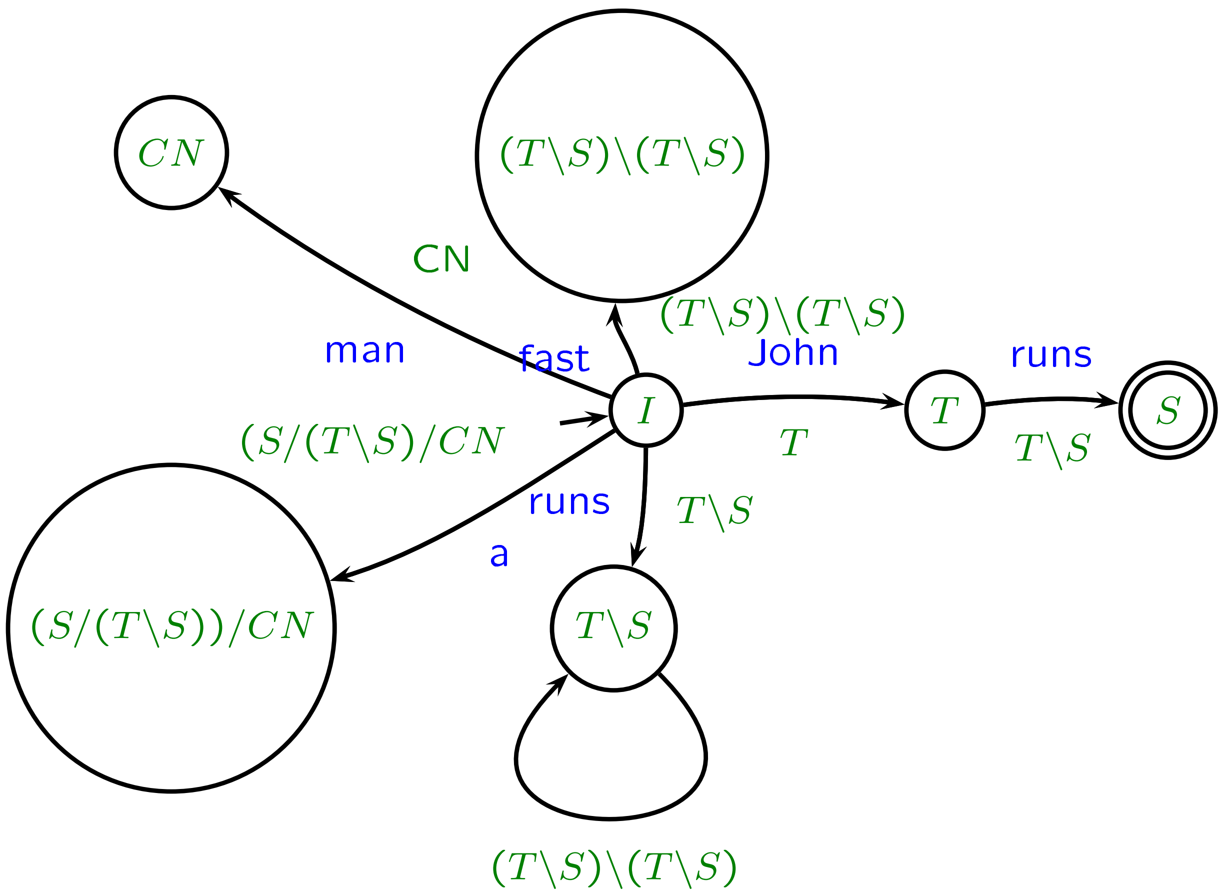
# Categorial Grammars and their properties

Links with Recursive Automata (Tellier06)

- A RA is like a Finite State Automaton except that transitions can be labelled by a state
- Using a transition labelled by a state $Q$ means producing $w \in L(Q)$
- There are two distinct kinds of RA :
  - the $RA_{FA}$-kind where the language $L(Q)$ of a state $Q$ is the set of strings from $Q$ to the final state
  - Every unidirect. $FA$ CG is strongly equivalent with a $RA_{FA}$
  - the $RA_{BA}$-kind where the language $L(Q)$ of a state $Q$ is the set of strings from the initial state to $Q$
  - Every unidirect. $BA$ CG is strongly equivalent with a $RA_{BA}$
- Every CG is equivalent with a pair $MRA = \langle RA_{FA}, RA_{FA} \rangle$



T
$S/(T\backslash S)$
$T\backslash S$
$T$
John
$(T\backslash S)\backslash(T\backslash S)$
$S$
$T\backslash S$
$F$
$(T\backslash S)\backslash(T\backslash S)$
runs
fast
man
CN
a
$S/(T\backslash S)$
$CN$
$(S/(T\backslash S))/CN$

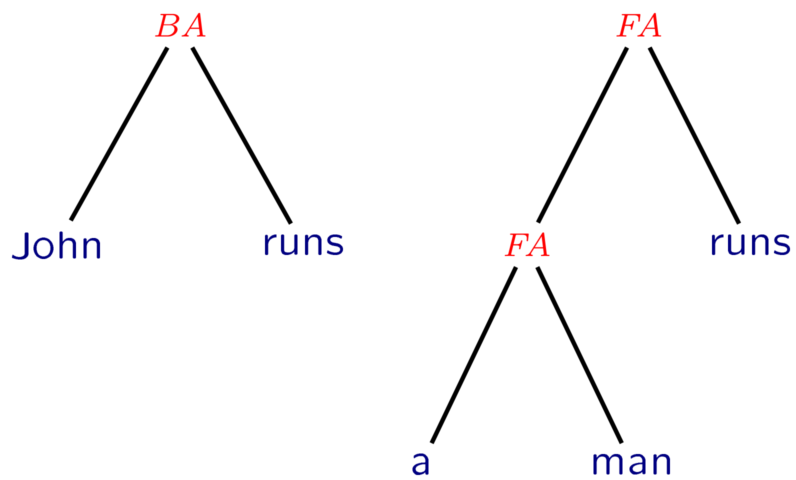



$CN$
$(T\backslash S)\backslash(T\backslash S)$
CN
$(T\backslash S)\backslash(T\backslash S)$
man
fast
John
runs
$I$
$T$
$S$
$(S/(T\backslash S)/CN$
$T$
$T\backslash S$
runs
$T\backslash S$
a
$(S/(T\backslash S))/CN$
$T\backslash S$
$(T\backslash S)\backslash(T\backslash S)$

# Learning CG by generalization

The BP (Buskowsky-Penn) algorithm

- target class : rigid CG, available data : strutural examples



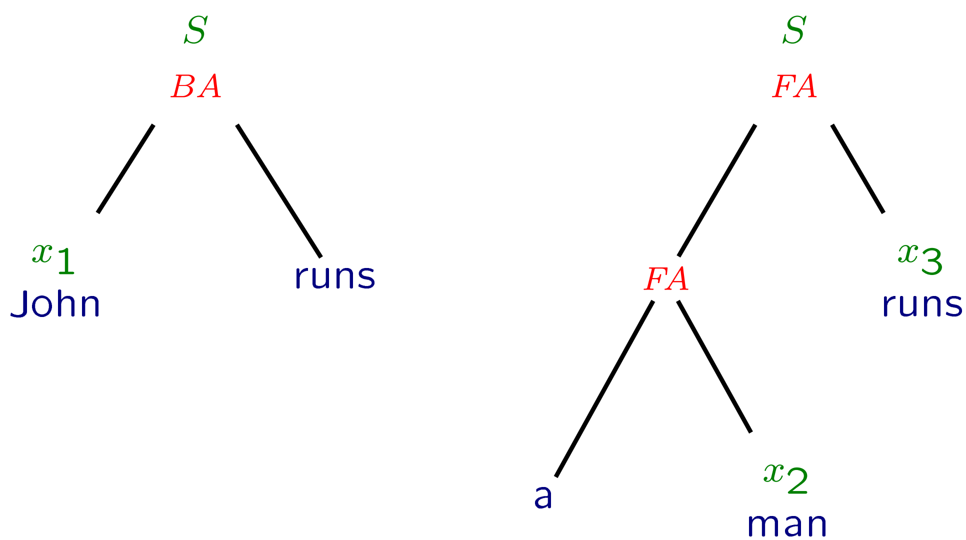

- algorithm :

1. introduce $S$ at the root and a distinct variable at each argument node
2. induce the other intermediate labels
3. collect the variable(s) associated with each word
4. try to unify them if there are several

# Learning CG by generalization

## The BP (Buskowsky-Penn) algorithm

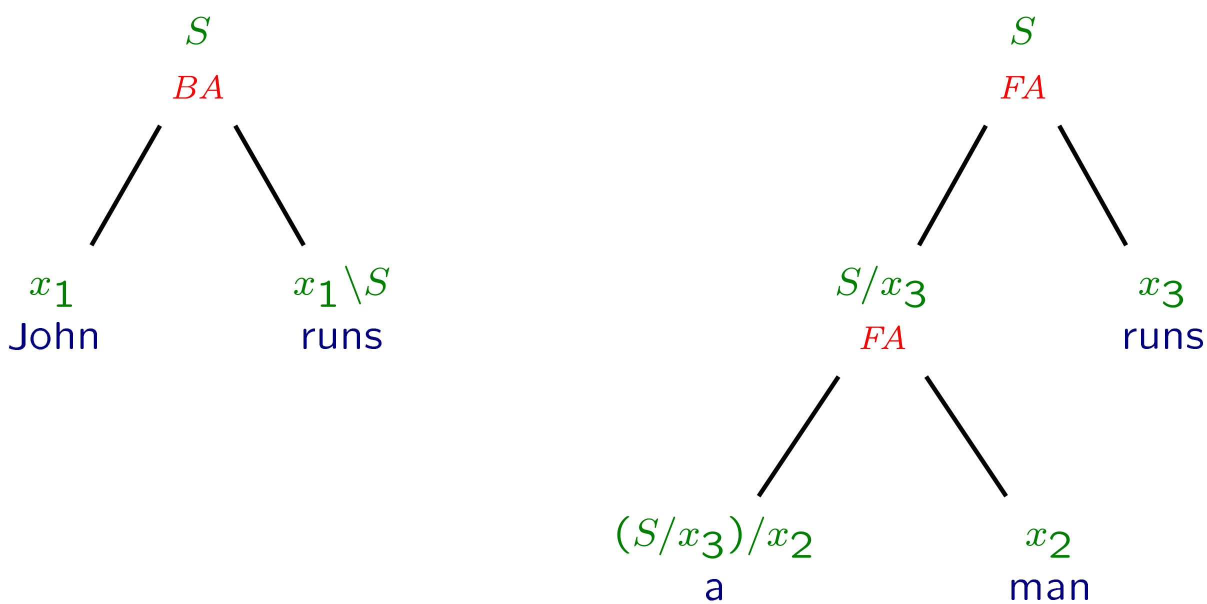
step 1 : introduce $S$ at the root and a distinct variable at each argument node



# Learning CG by generalization

The BP (Buskowsky-Penn) algorithm

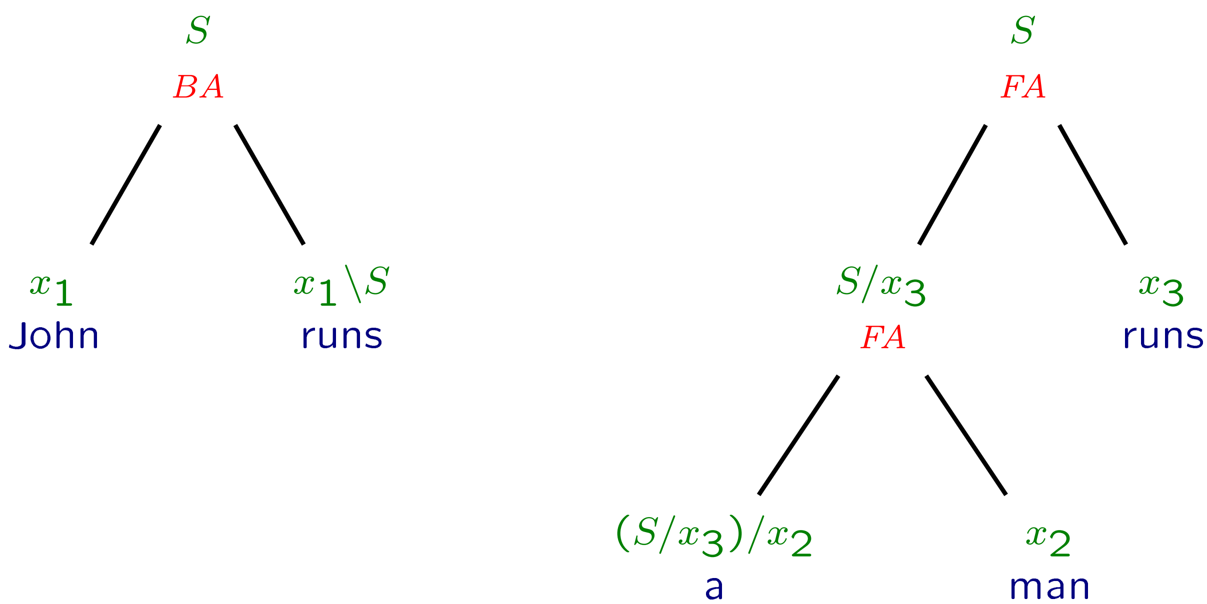
step 2 : induce the other intermediate labels



# Learning CG by generalization

The BP (Buskowsky-Penn) algorithm

step 3 : collect the variable(s) associated with each word

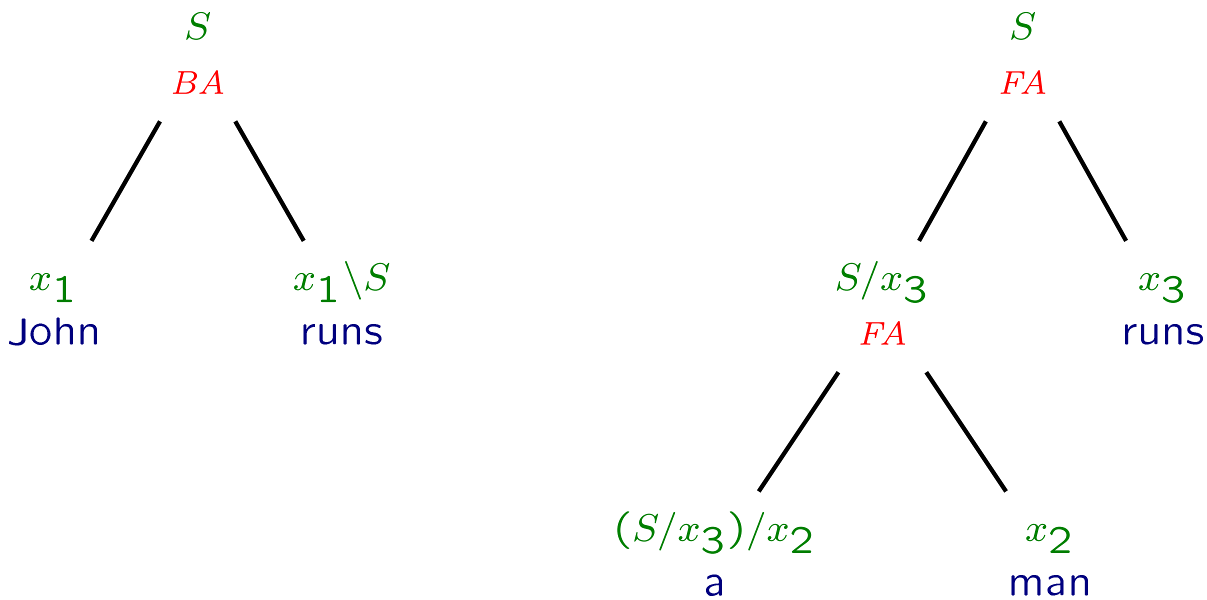



| word | category |
|---|---|
| John | $x_1$ |
| runs | $x_1\backslash S,\ x_3$ |
| man | $x_2$ |
| a | $(S/x_3)/x_2$ |

# Learning CG by generalization

The BP (Buskowsky-Penn) algorithm

step 4 : Try to unify them if there are several



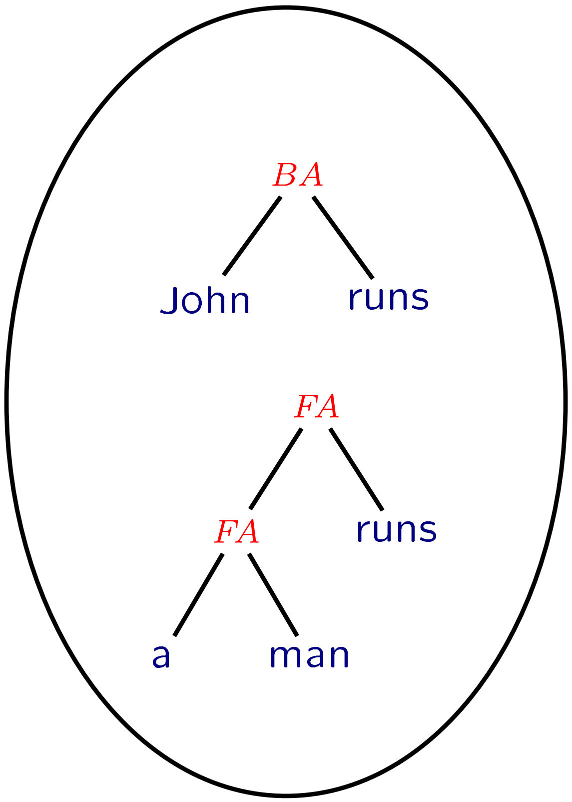

| word | category |
|---|---|
| John | $x_1$ |
| runs | $x_1 \backslash S = x_3$ |
| man | $x_2$ |
| a | $(S/(x_1 \backslash S))/x_2$ |

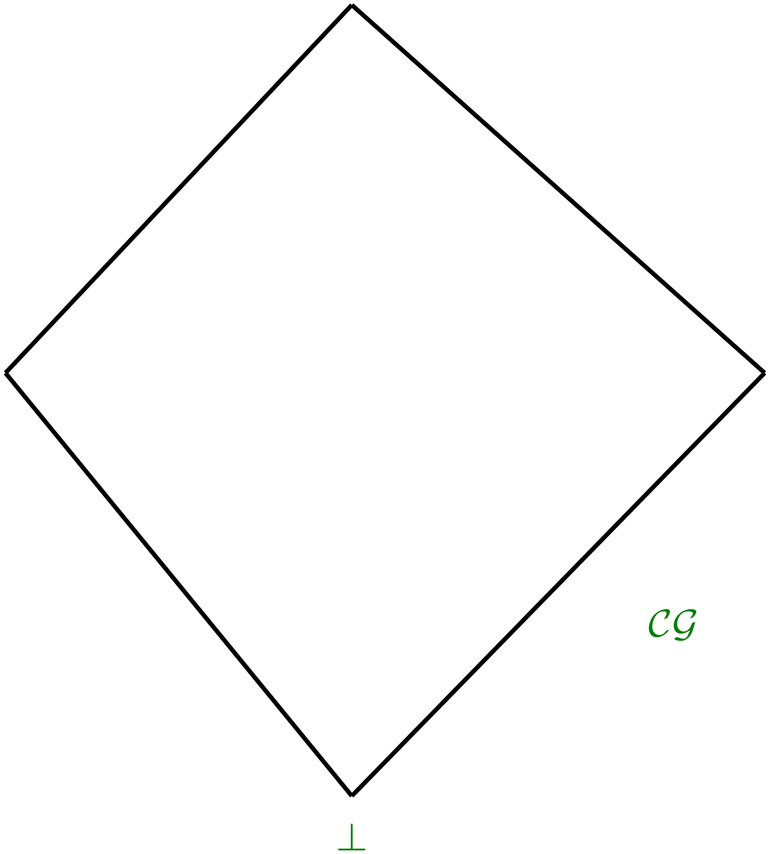
# Learning CG by generalization

General results

- this algorithm learns the class of rigid CGs from positive structural examples (Kanazawa 96, 98)
- it is linear in time, incremental...
- extensions are possible to learn
  - from strings (at the price of a combinatorial explosion)
  - the class of CG assigning at most $n$ category with each word (at the price of a combinatorial explosion)
- structural examples can be seen as coming from semantic information (Tellier 98)
- unifying variables can be seen as state and/or transition merges in the corresponding MRA (Tellier 06)

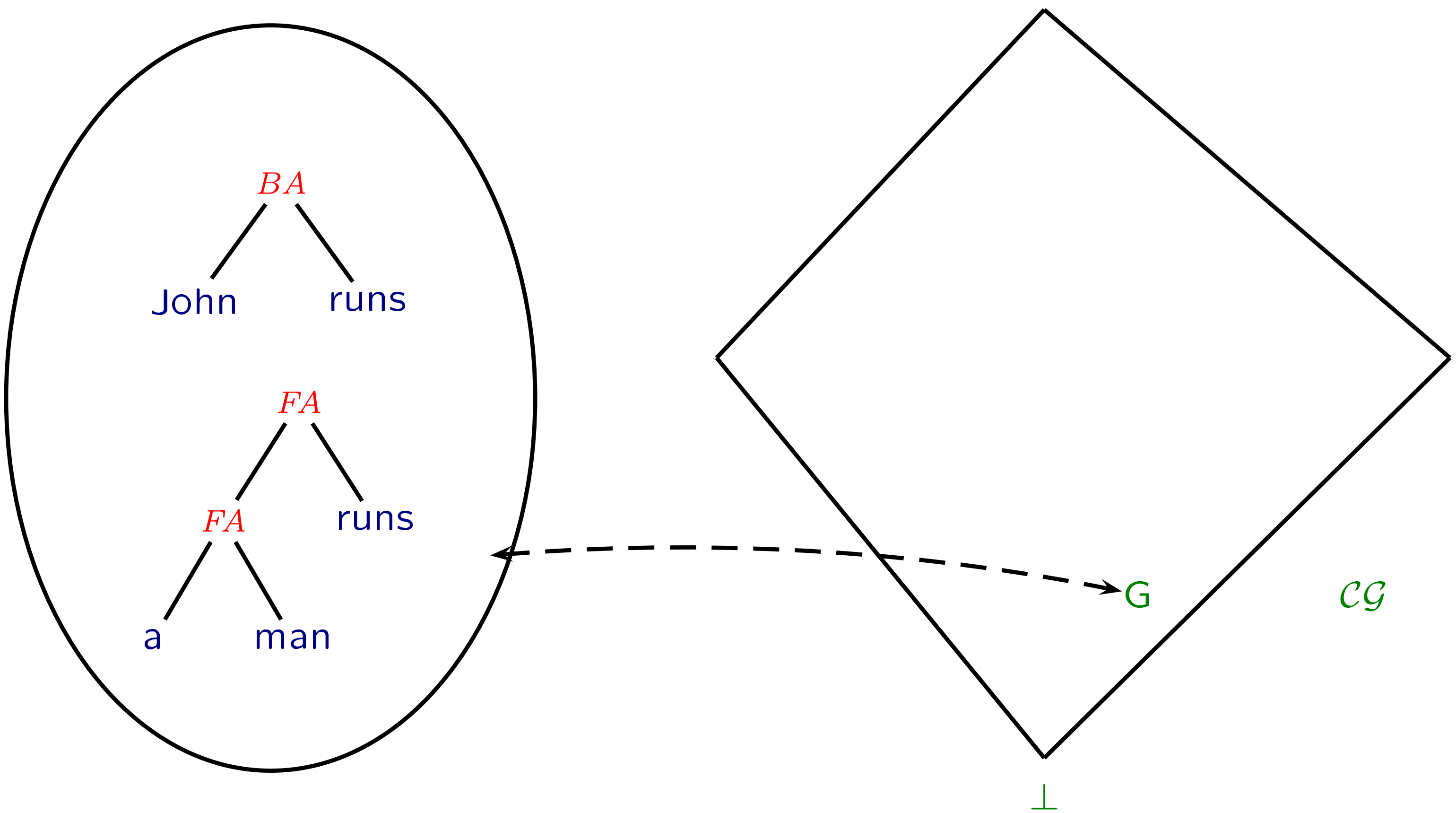
The BP (Buskowsky-Penn) algorithm : the very idea



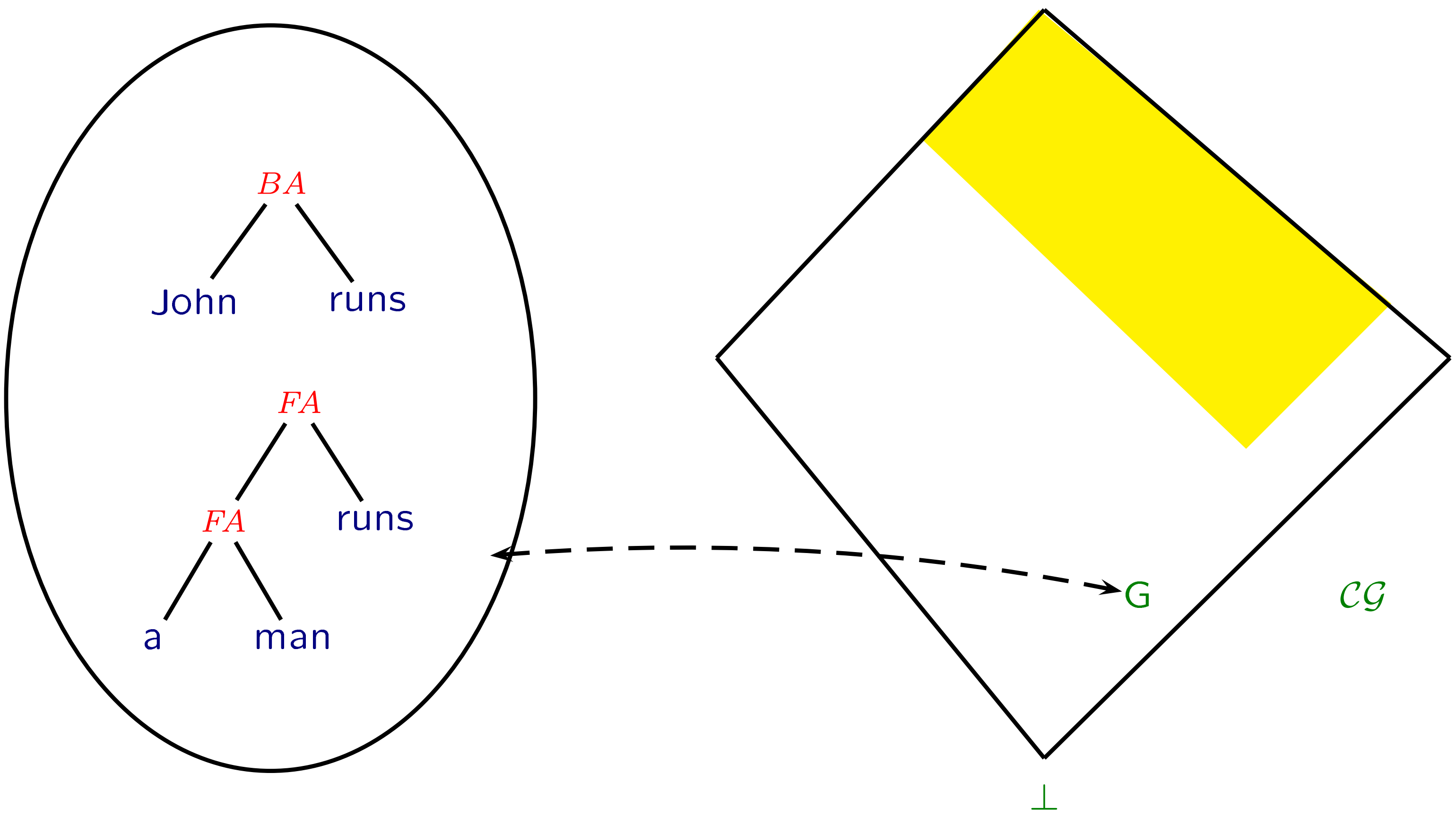



# Learning CG by generalization

Grammar specified by introducing variables

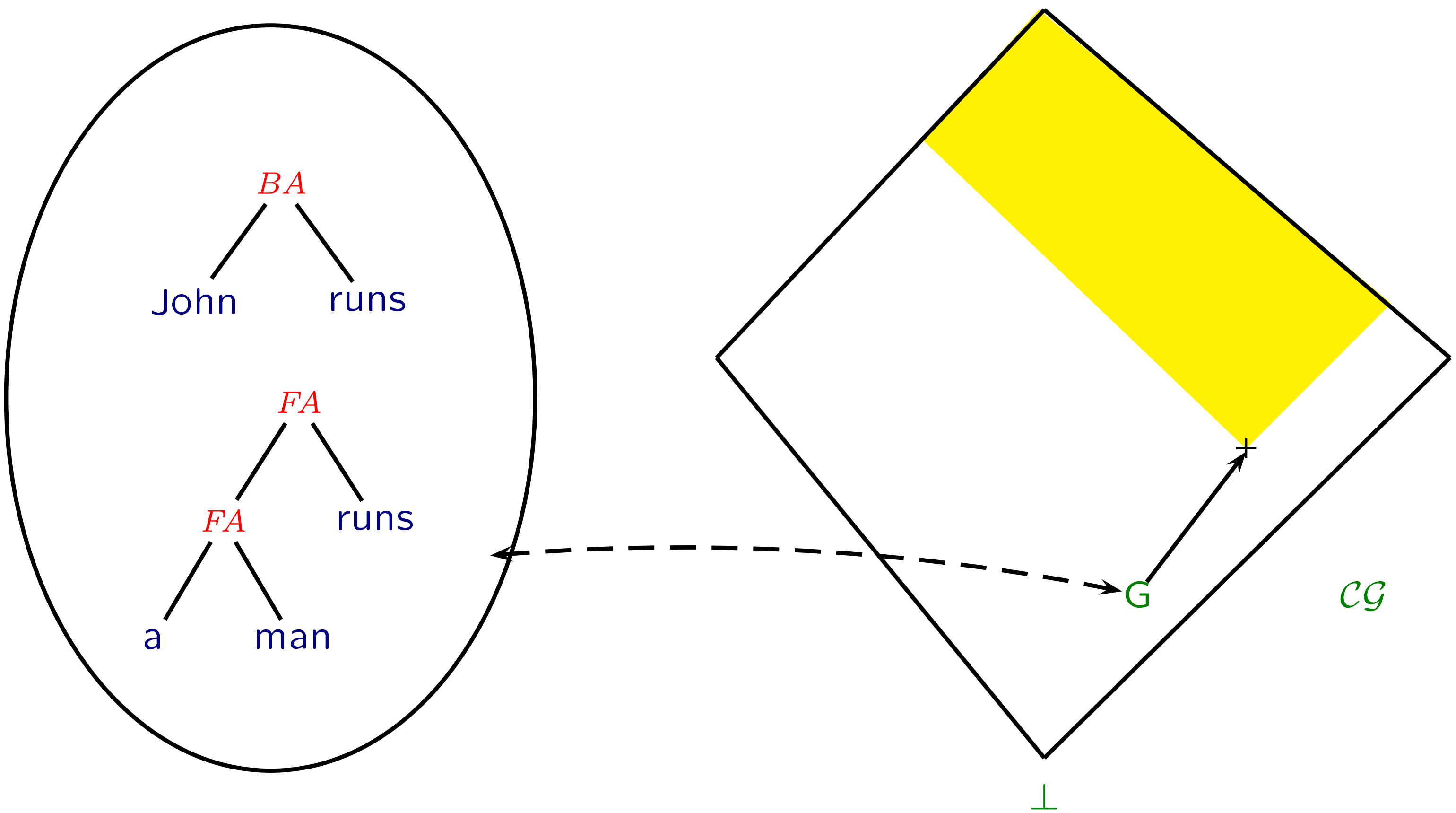


Subclass in which we search for the target



# Learning CG by generalization

Resulting grammar after unification



# Outline

1. Introduction : historical motivations
2. The learnability according to Gold
3. Categorial Grammars and their properties
4. Learning CG by generalization
5. Learning CG by specialization
6. Conclusion

# Learning CG by specialization

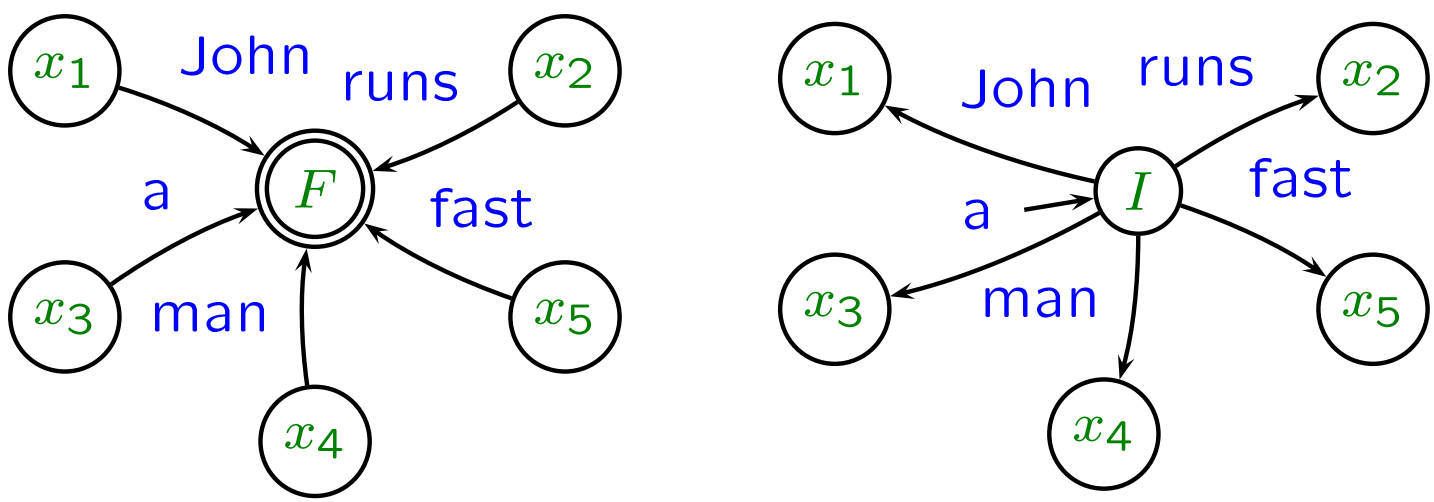
Algorithm (Moreau 04)

- target class : rigid CG, available data : strings
- algorithm :
  1. associate a distinct unique variable with each word
  2. for each sentence do
     - try to parse the sentence (CYK-like algorithm)
     - induce constraints on the variables
  3. output : (disjunctions of) set(s) of constraints, each set corresponding with a (set of) rigid grammar(s)

# Learning CG by specialization

Algorithm (Moreau 04)

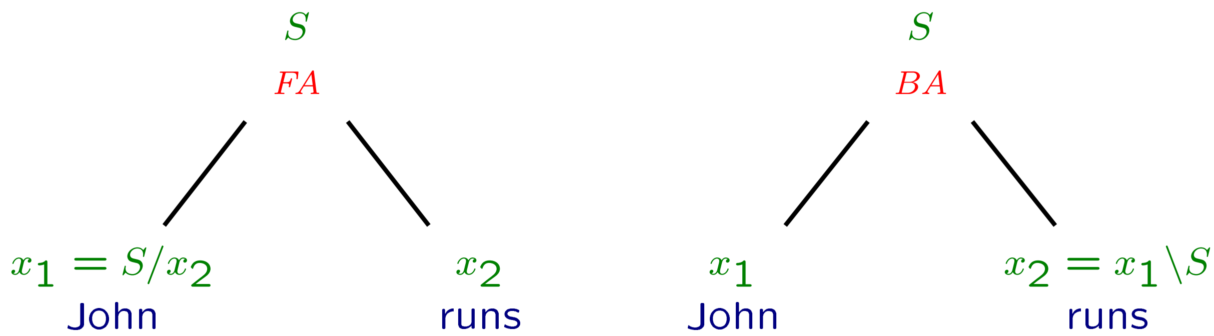
- input data : The set $D = \{$John runs, a man runs fast$\}$
- associate a distinct unique variable with each word :
  $\mathcal{A} = \{\langle$John, $x_1\rangle$, $\langle$runs, $x_2\rangle$, $\langle$a, $x_3\rangle$, $\langle$man, $x_4\rangle$, $\langle$fast, $x_5\rangle\}$
- for every rigid CG $G$, there exists a substitution from $\mathcal{A}$ to $G$
- $\mathcal{A}$ specifies the set of every rigid CGs
- $\mathcal{A}$ can also be represented by a $MRA = \langle RA_{FA}, RA_{BA}\rangle$ :



# Learning CG by specialization

## Algorithm (Moreau 04)

- the only two possible ways to parse “John runs” :




- to parse “a man runs fast” :
  - theoretically : $5 * 2^3 = 40$ distinct possible ways
  - but some couples of constraints are not compatible with the class of rigid grammars
- mainly operates state splits on the MRA
- main problem with this algo : combinatorial explosion
- to limit it : initial knowledge in the form of known assignments

# Learning CG by specialization

Learning From Typed Examples (Dudau, Tellier &Tommasi 01)

- cognitive hypothesis : lexical semantics is learned before syntax
- formalization : words are given with their (Montagovian) semantic type
- types derive from categories by a homomorphism
- recall : $h(T) = e,\ h(S) = t,\ h(CN) = \langle e, t \rangle$ and $h(A/B) = h(B \backslash A) = \langle h(B), h(A) \rangle$
- input data : typed sentences are of the form

| John | runs |
| --- | --- |
| $e$ | $\langle e, t \rangle$ |

| a | man | runs | fast |
| --- | --- | --- | --- |
| $\langle\langle e, t \rangle, \langle\langle e, t \rangle, t \rangle\rangle$ | $\langle e, t \rangle$ | $\langle e, t \rangle$ | $\langle\langle e, t \rangle, \langle e, t \rangle\rangle$ |

# Learning CG by specialization

Learning From Typed Examples (Dudau, Tellier &Tommasi 01)

- target class : the set of CGs such that every distinct category assigned to the same word gives a distinct type
- $\forall \langle v, C_1 \rangle, \langle v, C_2 \rangle \in G$, $C_1 \neq C_2 \Longrightarrow h(C_1) \neq h(C_2)$
- Theorem (Dudau, Tellier & Tommasi 03) : for every CF-language, there exists $G, h$ satisfying this condition
- learning algorithm (Dudau, Tellier &Tommasi 01)

  1. introduce variables to represent the class
  2. for each sentence
     - try to parse the sentence (CYK-like)
     - induce constraints on the variables
  3. output : (disjunctions) of set(s) of contraint(s), each being represented by a least general grammar

## Learning From Typed Examples (Dudau, Tellier &Tommasi 01)

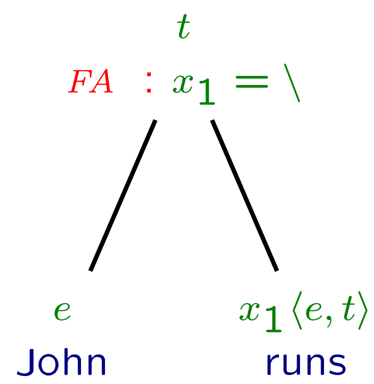
step 1 : introduce variables to represent the class : a distinct one whose possible values are / or \ in front of every subtype

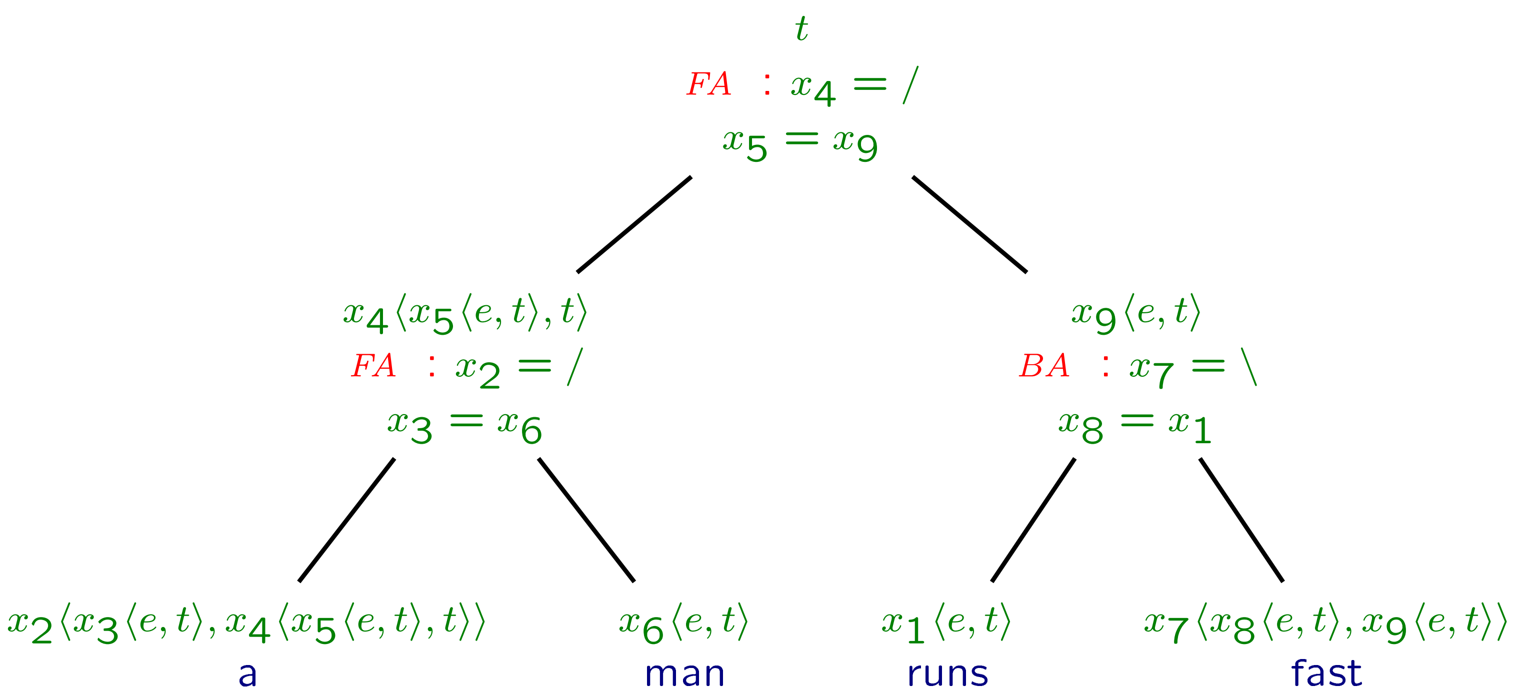
| John | runs |
|---|---|
| $e$ | $x_1\langle e,t\rangle$ |

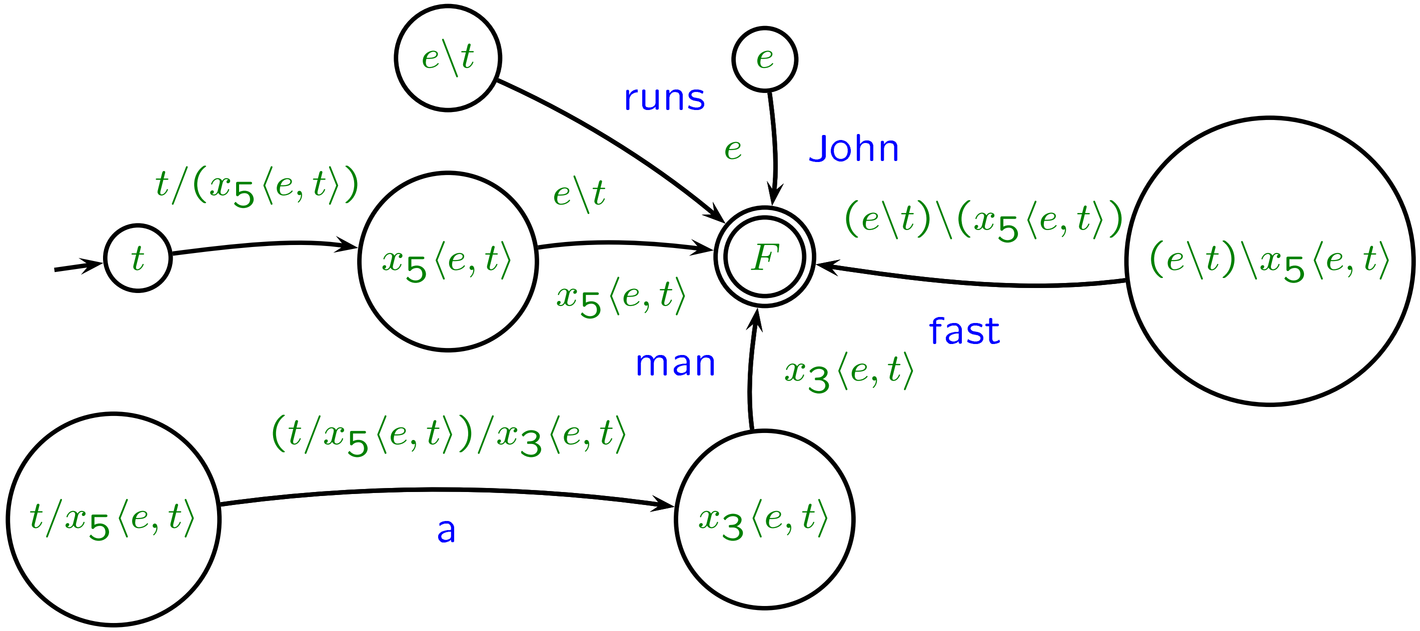
| a | man | runs | fast |
|---|---|---|---|
| $x_2\langle x_3\langle e,t\rangle, x_4\langle x_5\langle e,t\rangle, t\rangle\rangle$ | $x_6\langle e,t\rangle$ | $x_1\langle e,t\rangle$ | $x_7\langle x_8\langle e,t\rangle, x_9\langle e,t\rangle\rangle$ |

# Learning CG by specialization

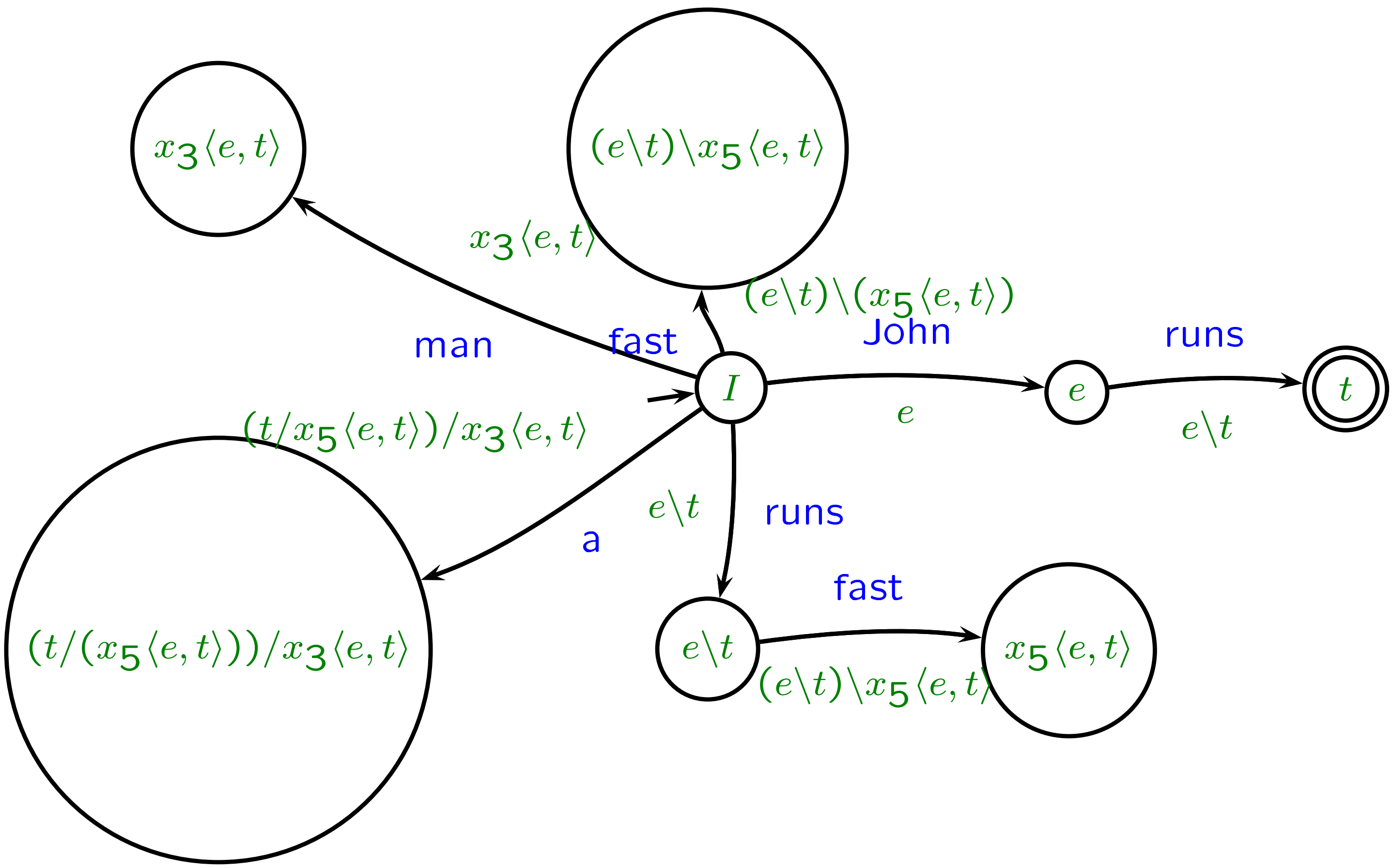
step 2 : for each sentence, try to parse and induce constraints





$e\backslash t$
$e$
runs
$e$
John
$t/(x_5\langle e,t\rangle)$
$e\backslash t$
$t$
$x_5\langle e,t\rangle$
$F$
$(e\backslash t)\backslash(x_5\langle e,t\rangle)$
$(e\backslash t)\backslash x_5\langle e,t\rangle$
$x_5\langle e,t\rangle$
fast
man
$x_3\langle e,t\rangle$
$(t/x_5\langle e,t\rangle)/x_3\langle e,t\rangle$
$t/x_5\langle e,t\rangle$
$x_3\langle e,t\rangle$
a

$x_3\langle e,t\rangle$
$(e\backslash t)\backslash x_5\langle e,t\rangle$
$x_3\langle e,t\rangle$
$(e\backslash t)\backslash(x_5\langle e,t\rangle)$
man
fast
John
runs
$I$
$e$
$t$
$(t/x_5\langle e,t\rangle)/x_3\langle e,t\rangle$
$e$
$e\backslash t$
$e\backslash t$
runs
a
fast
$(t/(x_5\langle e,t\rangle))/x_3\langle e,t\rangle$
$e\backslash t$
$x_5\langle e,t\rangle$
$(e\backslash t)\backslash x_5\langle e,t\rangle$

## Sum-up

- combination of state splits ($x_1 = \backslash$) and state merges ($x_3 = x_6$)
- types contain in themselves where splits are possible
- not every (complex) state can be merged : only those that are unifiable in the sense of (Coste & alii 2004)
- types reduce the combinatorial explosion of possible splits and help to converge to the correct solution quicker
- the starting point is either a lower ound or an upper bound of the target (linked to it by a morphism)

| vocabulary | Moreau's initial assigment | target category | pre-treated type |
|---|---|---|---|
| John | $x_1$ | $T$ | $e$ |
| a | $x_2$ | $(S/(T\backslash S))/CN$ | $x_2\langle x_3\langle e,t\rangle, x_4\langle x_5\langle e,t\rangle, t\rangle\rangle$ |
| man | $x_3$ | $CN$ | $x_6\langle e,t\rangle$ |
| runs | $x_4$ | $T\backslash S$ | $x_1\langle e,t\rangle$ |

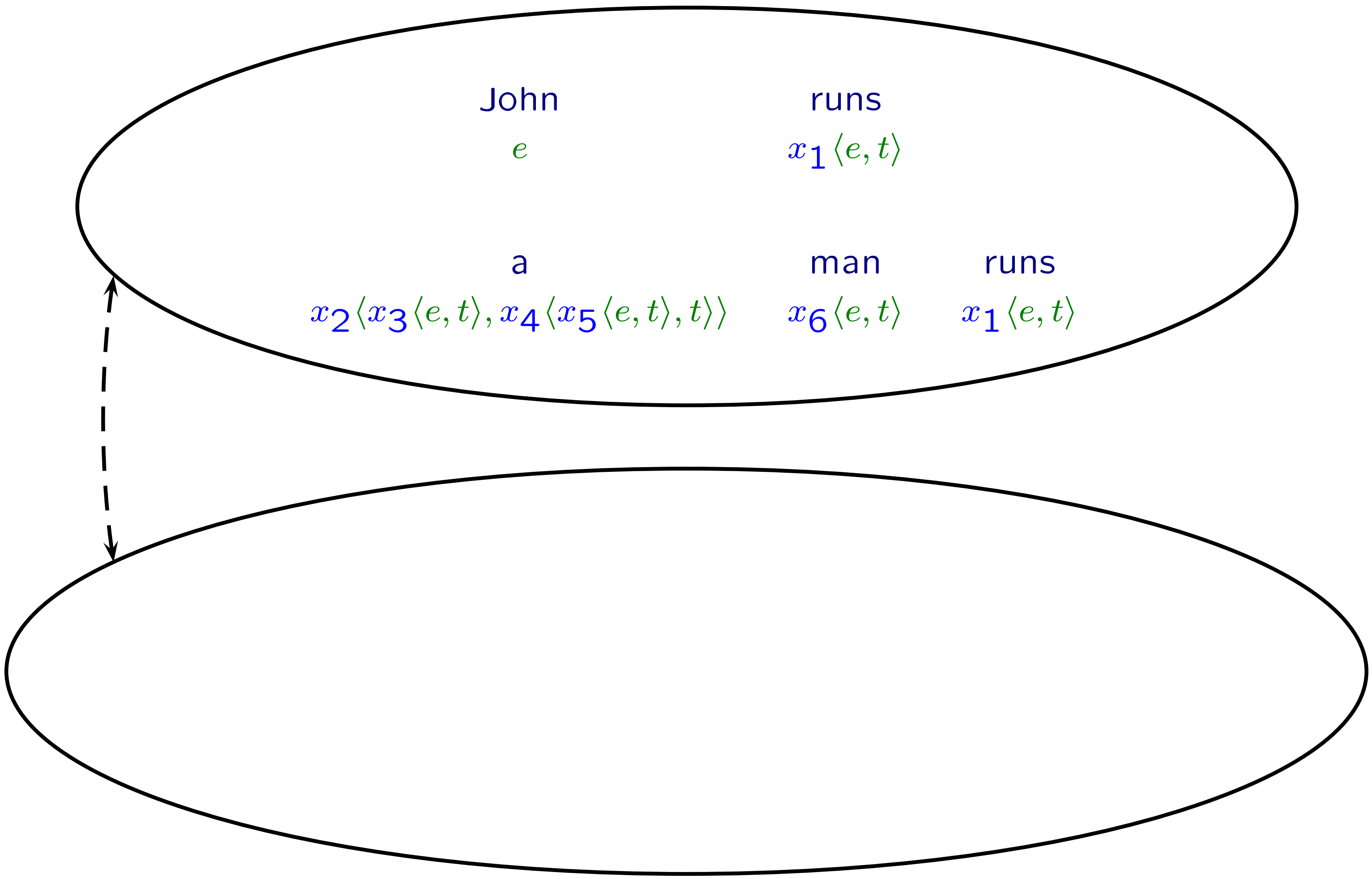
# Learning CG by specialization

## Learning from Typed Examples : the very idea

| | | |
|---|---|---|
| John | runs | |
| $e$ | $\langle e,t \rangle$ | |
| a | man | runs |
| $\langle\langle e,t \rangle, \langle\langle e,t \rangle, t \rangle\rangle$ | $\langle e,t \rangle$ | $\langle e,t \rangle$ |

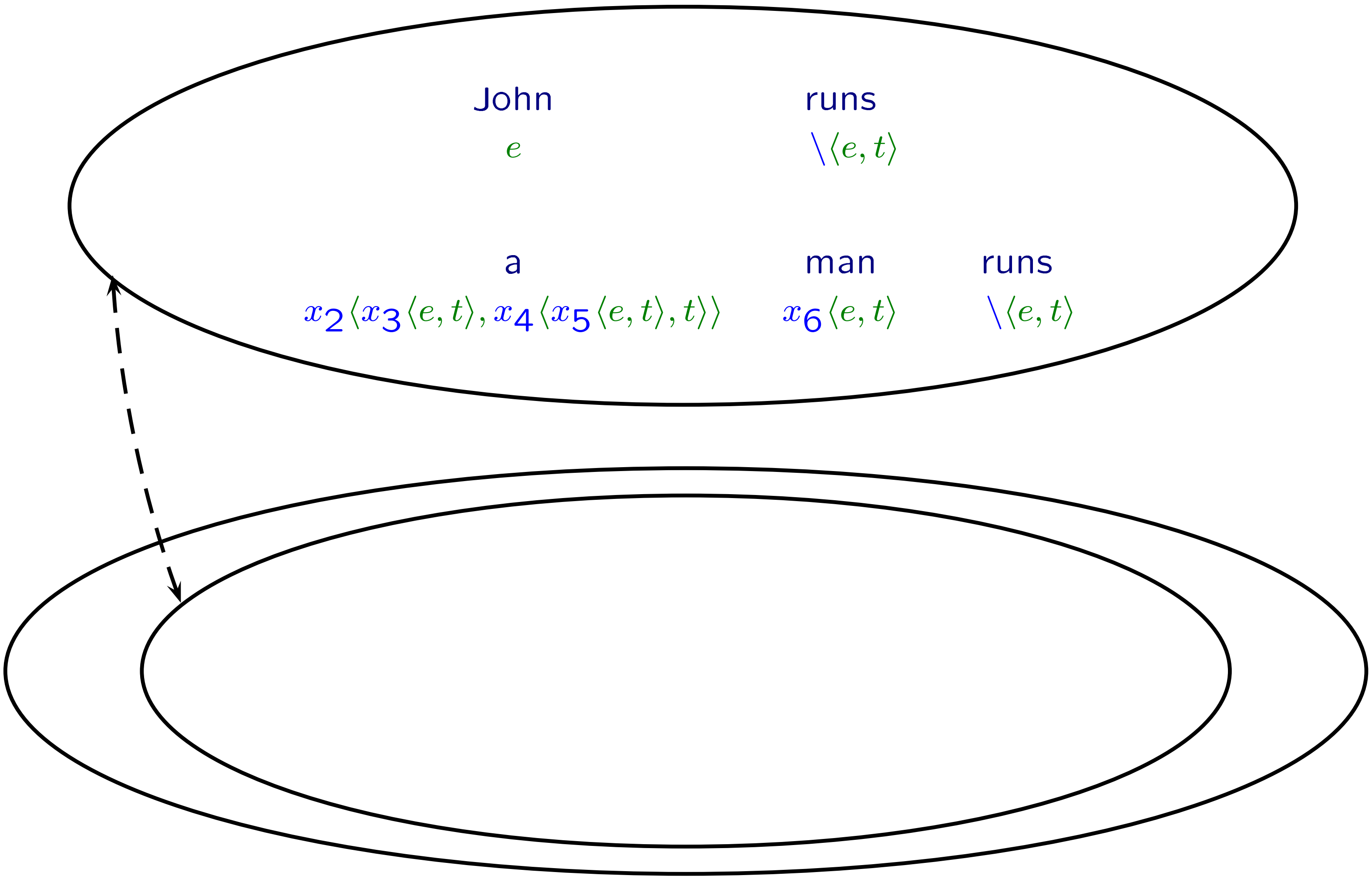
# Learning CG by specialization

## Learning from Typed Examples : introduce variables



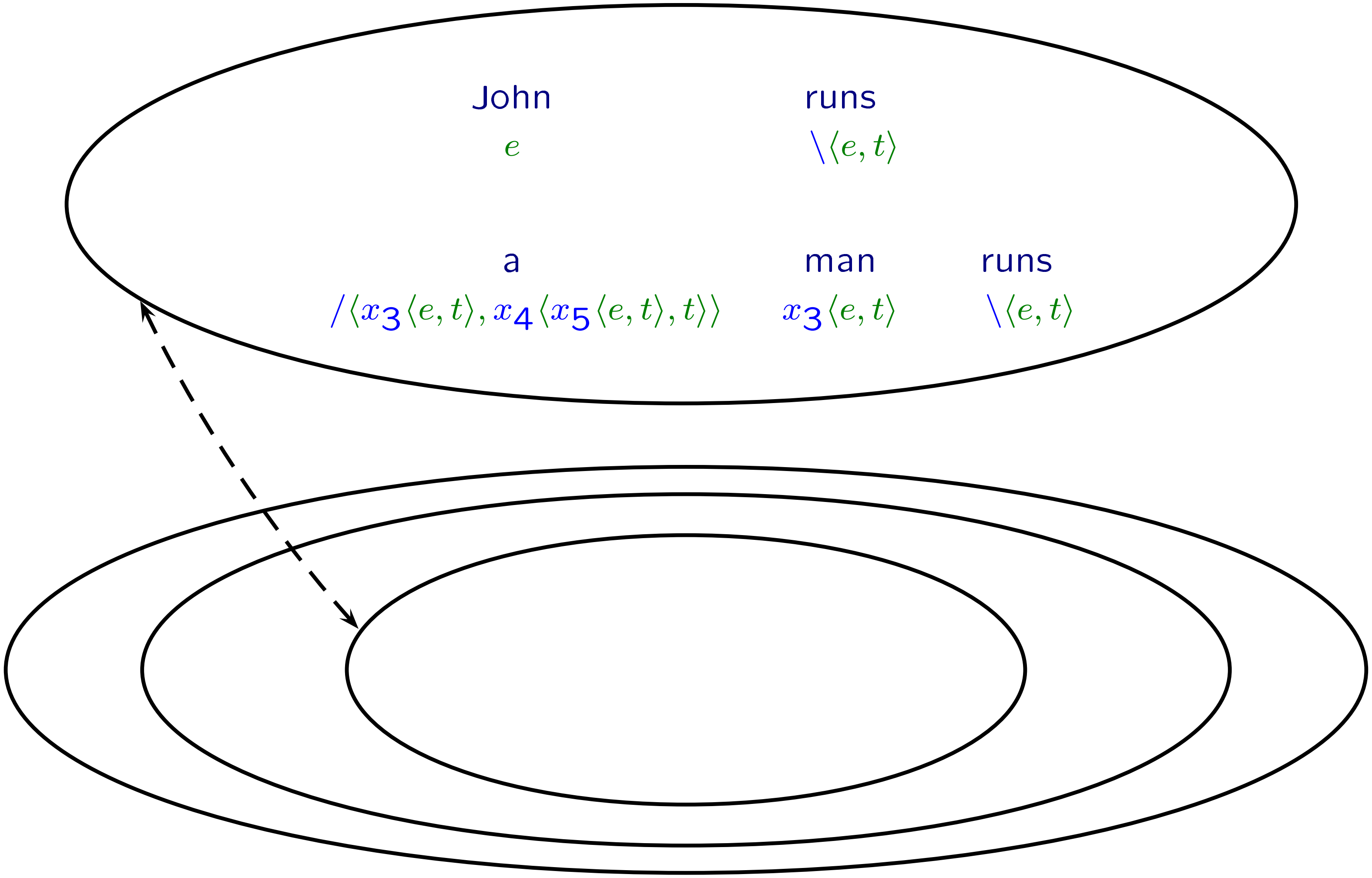
# Learning CG by specialization

Constraint learned : $x_1 = \backslash$



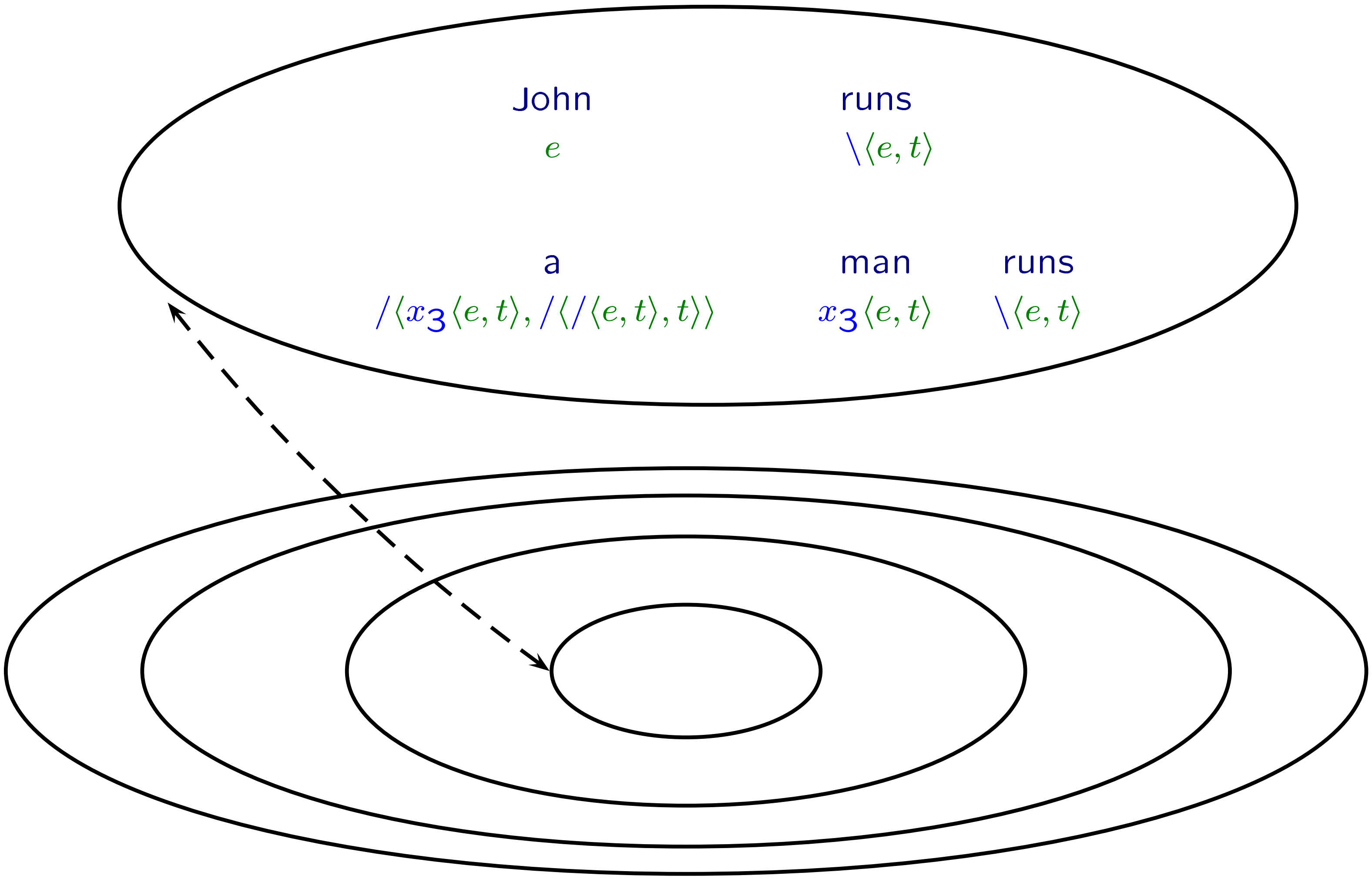
# Learning CG by specialization

Constraint learned : $x_2 = /,\ x_3 = x_6$



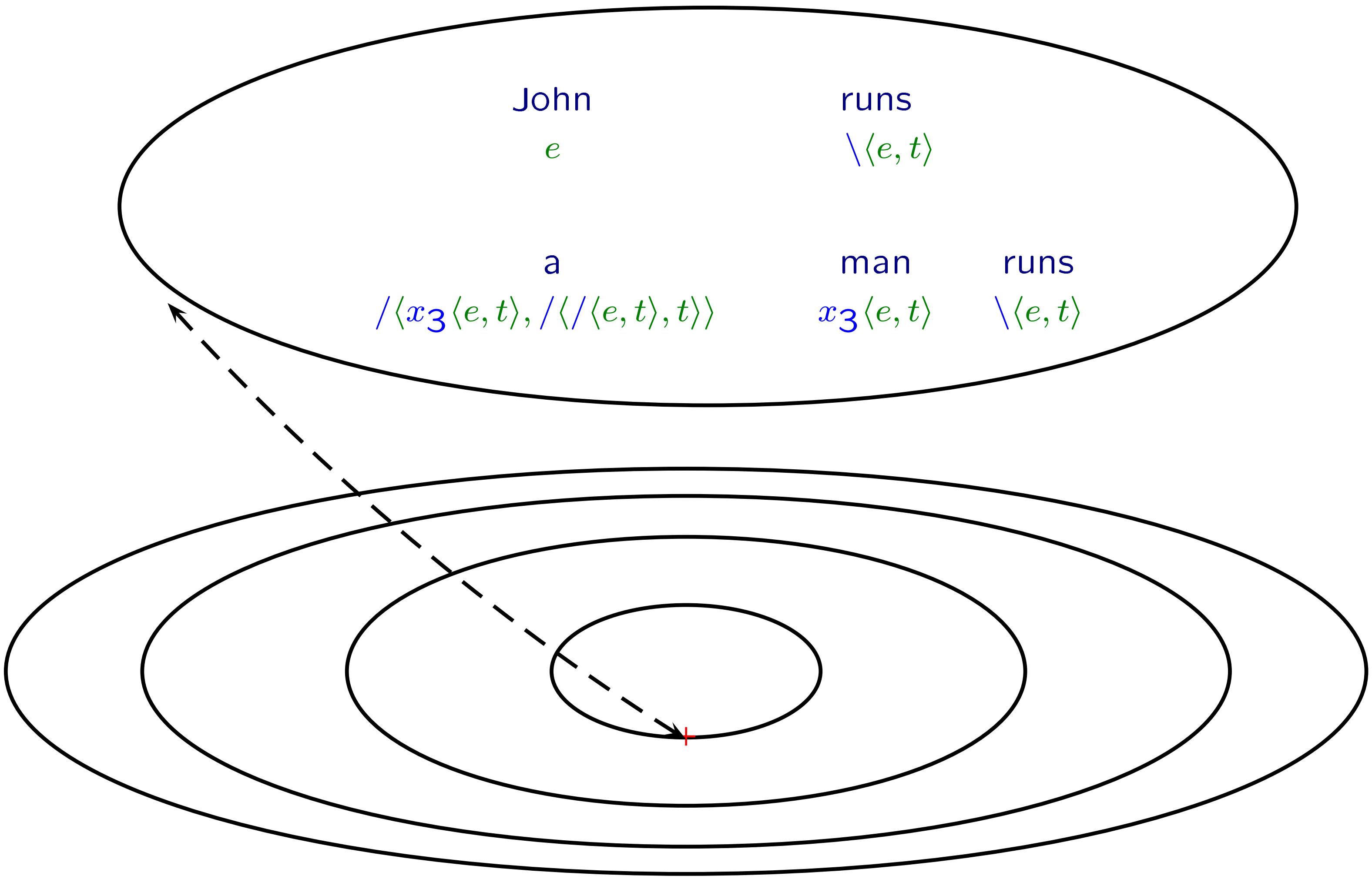
# Learning CG by specialization

Constraint learned : $x_4 = /,\ x_5 = x_1 = \backslash$



# Learning CG by specialization

Output : the least general grammar in the set



# Learning CG by specialization

Sum-up

- learnability from typed data "in the limit" assured in the new subclass
- good properties of the new subclass
- types can be considered as lexical semantic information
- the set of possible grammars decreases while the resulting grammar(s) generalize(s)

  $\Longrightarrow$ strategy coherent with natural language learning
- a prototype has been implemented and tested on real data

# Outline

1. Introduction : historical motivations
2. The learnability according to Gold
3. Categorial Grammars and their properties
4. Learning CG by generalization
5. Learning CG by specialization
6. Conclusion

General critics on the approach

- Gold's criterion too rudimentary
- strong cognitive asumptions about symbolic internal representations (not realistic)
- formal approaches are too sensistive to "noises" (errors, mistakes...)
- mainly theoretical results (algorithms not tractable)

Main achievements

- connexion established between various domains, pluridisciplinarity
- formalizes the conditions of possibility of learning
- semantics considered as providing structures
  - for generalization : in the form of structural examples
  - for specialization : in the form of structural lexical types