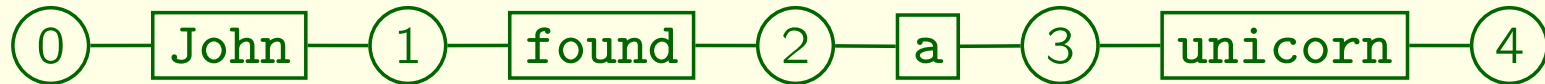


# **Parsing and Generation as Datalog Queries**

**Makoto Kanazawa**

**National Institute of Informatics**

# CFG recognition/parsing as Datalog query evaluation



```
S(i,j) :- NP(i,k), VP(k,j).  
VP(i,j) :- V(i,k), NP(k,j).  
NP(i,j) :- Det(i,k), N(k,j).  
NP(i,j) :- John(i,j).  
V(i,j) :- found(i,j).  
Det(i,j) :- a(i,j).  
N(i,j) :- unicorn(i,j).
```

*Datalog program*

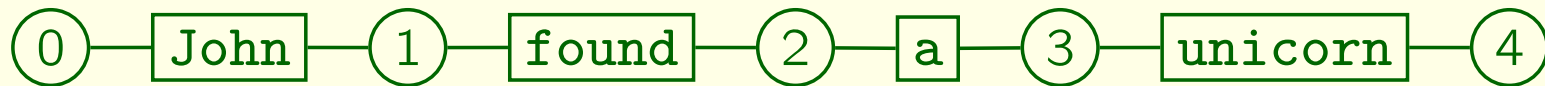
```
John(0,1).  
found(1,2).  
a(2,3).  
unicorn(3,4).
```

*database*

```
?- S(0,4).
```

*query*

# CFG recognition/parsing as Datalog query evaluation



```
S(i,j) :- NP(i,k), VP(k,j).  
VP(i,j) :- V(i,k), NP(k,j).  
NP(i,j) :- Det(i,k), N(k,j).  
NP(i,j) :- John(i,j).  
V(i,j) :- found(i,j).  
Det(i,j) :- a(i,j).  
N(i,j) :- unicorn(i,j).
```

*Datalog program*

```
John(0,1).  
found(1,2).  
a(2,3).  
unicorn(3,4).
```

*database*

```
?- S(0,4).
```

*query*

- Polynomial-time algorithms
  - CYK  $\approx$  seminaive bottom-up evaluation
  - Earley  $\approx$  magic-sets rewriting

# Extending Datalog representation

CFG recognition/parsing



recognition/parsing for  
grammars with  
“context-free” derivations

(MC)TAG

(P)MCFG

surface realization for  
CFGs with  
Montague semantics  
(suitably restricted)

# Extending Datalog representation

CFG recognition/parsing



recognition/parsing for  
grammars with  
“context-free” derivations  
(MC)TAG  
(P)MCFG

surface realization for  
CFGs with  
Montague semantics  
(suitably restricted)

- Polynomial-time algorithms
  - CYK  $\approx$  seminaive bottom-up evaluation
  - Earley  $\approx$  magic-sets rewriting

# Extending Datalog representation

CFG recognition/parsing

recognition/parsing for  
grammars with  
“context-free” derivations  
(MC)TAG  
(P)MCFG

surface realization for  
CFGs with  
Montague semantics  
(suitably restricted)

- Polynomial-time algorithms

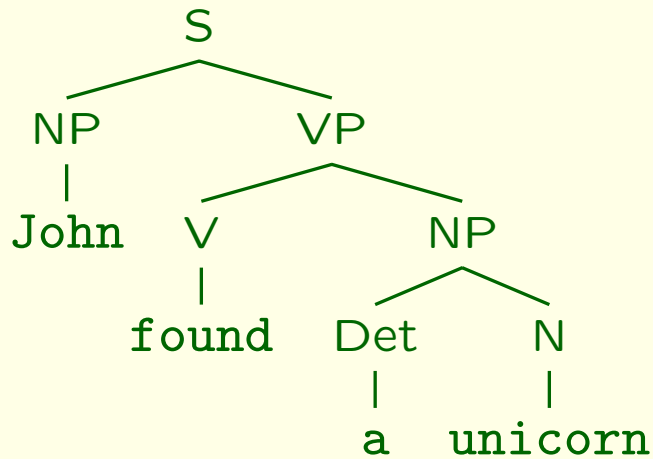
- CYK  $\approx$  seminaive bottom-up evaluation
- Earley  $\approx$  magic-sets rewriting

- Complexity

- Polynomial-sized derivation tree  $\Rightarrow$  LOGCFL

$AC^0 \subseteq NC^1 \subseteq L \subseteq SL \subseteq NL \subseteq LOGCFL \subseteq AC^1 \subseteq NC^2 \subseteq P \subseteq NP$

# Context-free grammar with Montague semantics



$$S(X_1 X_2) \rightarrow NP(X_1) \ VP(X_2)$$

$$VP(\lambda x. X_2(\lambda y. X_1 y x)) \rightarrow V(X_1) \ NP(X_2)$$

$$V(\lambda y x. X_2(X_1 y x)(X_3 y x)) \rightarrow V(X_1) \ Conj(X_2) \ V(X_3)$$

$$NP(X_1 X_2) \rightarrow Det(X_1) \ N(X_2)$$

$$NP(\lambda u. u \mathbf{John}^e) \rightarrow \mathbf{John}$$

$$V(\mathbf{find}^{e \rightarrow e \rightarrow t}) \rightarrow \mathbf{found}$$

$$V(\mathbf{catch}^{e \rightarrow e \rightarrow t}) \rightarrow \mathbf{caught}$$

$$Conj(\Lambda^{t \rightarrow t \rightarrow t}) \rightarrow \mathbf{and}$$

$$Det(\lambda u v. \exists^{(e \rightarrow t) \rightarrow t}(\lambda y. \Lambda^{t \rightarrow t \rightarrow t}(u y)(v y))) \rightarrow \mathbf{a}$$

$$N(\mathbf{unicorn}^{e \rightarrow t}) \rightarrow \mathbf{unicorn}$$

$$(\lambda u. u \mathbf{John})(\lambda x. (\lambda u v. \exists(\lambda y. \Lambda(u y)(v y))) \mathbf{unicorn} (\lambda y. \mathbf{find} y x))$$

$$\rightarrow_{\beta} \exists(\lambda y. \Lambda(\mathbf{unicorn} y)(\mathbf{find} y \mathbf{John}))$$

$$\approx \exists y(\mathbf{unicorn}(y) \wedge \mathbf{find}(\mathbf{John}, y))$$

*logical form*

# Semantic interpretation and surface realization

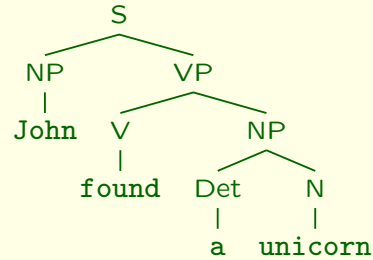
## Semantic interpretation

*string*

John found a unicorn

$O(n^3)$

*derivation tree*



easy

*logical form*

$\exists(\lambda y. \Lambda(\text{unicorn } y)(\text{find } y \text{ John}))$

## Surface realization (tactical generation)

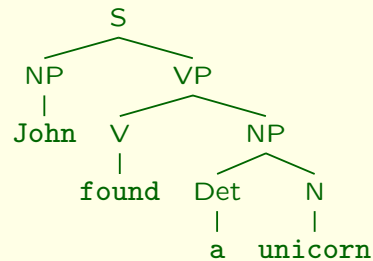
*logical form*

$\exists(\lambda y. \Lambda(\text{unicorn } y)(\text{find } y \text{ John}))$

$\Rightarrow$

?

*derivation tree*



$\Rightarrow$

easy

*string*

John found a unicorn

# Semantic interpretation and surface realization

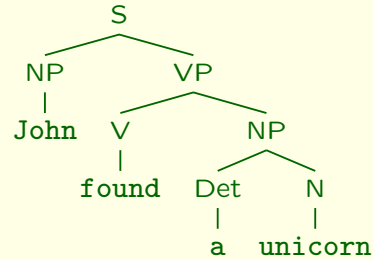
## Parsing (of input string)

*string*

John found a unicorn

$\Rightarrow$   
 $O(n^3)$

*derivation tree*



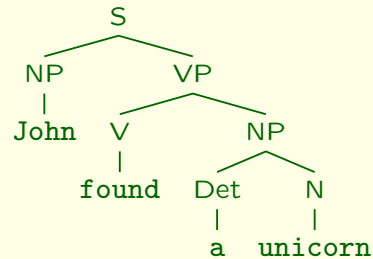
## Parsing of input logical form

*logical form*

$\exists(\lambda y.\Lambda(\text{unicorn } y)(\text{find } y \text{ John}))$

$\Rightarrow$   
?

*derivation tree*



# Semantic interpretation and surface realization

## Recognition (of input string)

*string*

John found a unicorn

$\implies$

*grammatical?*

yes/no

$O(n^3)$

## Recognition of input logical form

*logical form*

$\exists(\lambda y.\Lambda(\text{unicorn } y)(\text{find } y \text{ John}))$

$\implies$

*surface realizable?*

yes/no

?

Recognition  $\approx$  Parsing

## My approach to surface realization

- Exact generation, not taking into account logical equivalence.

$$\mathbf{man\ John} \Leftrightarrow \exists y. (\mathbf{man\ } y \wedge y = \mathbf{John})$$

## My approach to surface realization

- Exact generation, not taking into account logical equivalence.

$$\mathbf{man\ John} \Leftrightarrow \exists y. (\mathbf{man\ } y \wedge y = \mathbf{John})$$

- The semantic representation language is that of **higher-order**  $\lambda$ -terms, and the computation of meaning involves  **$\beta$ -reduction**.
- More faithful to standard linguistic analysis than previous approaches.

# Synchronous grammar view

$S(X_1X_2) \rightarrow NP(X_1) VP(X_2)$   
 $VP(\lambda x.X_2(\lambda y.X_1yx)) \rightarrow V(X_1) NP(X_2)$   
 $V(\lambda yx.X_2(X_1yx)(X_3yx)) \rightarrow V(X_1) Conj(X_2) V(X_3)$   
 $NP(X_1X_2) \rightarrow Det(X_1) N(X_2)$   
 $NP(\lambda u.u \mathbf{John}^e) \rightarrow \mathbf{John}$   
 $V(\mathbf{find}^{e \rightarrow e \rightarrow t}) \rightarrow \mathbf{found}$   
 $V(\mathbf{catch}^{e \rightarrow e \rightarrow t}) \rightarrow \mathbf{caught}$   
 $Conj(\Lambda^{t \rightarrow t \rightarrow t}) \rightarrow \mathbf{and}$   
 $Det(\lambda uv.\exists^{(e \rightarrow t) \rightarrow t}(\lambda y.\Lambda^{t \rightarrow t \rightarrow t}(uy)(vy))) \rightarrow \mathbf{a}$   
 $N(\mathbf{unicorn}^{e \rightarrow t}) \rightarrow \mathbf{unicorn}$

*string grammar*

$S \rightarrow NP VP$   
 $VP \rightarrow V NP$   
 $V \rightarrow V Conj V$   
 $NP \rightarrow Det N$   
 $NP \rightarrow \mathbf{John}$   
 $V \rightarrow \mathbf{found}$   
 $V \rightarrow \mathbf{caught}$   
 $Conj \rightarrow \mathbf{and}$   
 $Det \rightarrow \mathbf{a}$   
 $N \rightarrow \mathbf{unicorn}$

*$\lambda$ -term grammar*

$S(X_1X_2) :- NP(X_1), VP(X_2).$   
 $VP(\lambda x.X_2(\lambda y.X_1yx)) :- V(X_1), NP(X_2).$   
 $V(\lambda yx.X_2(X_1yx)(X_3yx)) :- V(X_1), Conj(X_2), V(X_3).$   
 $NP(X_1X_2) :- Det(X_1), N(X_2).$   
 $NP(\lambda u.u \mathbf{John}^e).$   
 $V(\mathbf{find}^{e \rightarrow e \rightarrow t}).$   
 $V(\mathbf{catch}^{e \rightarrow e \rightarrow t}).$   
 $Conj(\Lambda^{t \rightarrow t \rightarrow t}).$   
 $Det(\lambda uv.\exists^{(e \rightarrow t) \rightarrow t}(\lambda y.\Lambda^{t \rightarrow t \rightarrow t}(uy)(vy))).$   
 $N(\mathbf{unicorn}^{e \rightarrow t}).$

# Context-free $\lambda$ -term grammars (CFLGs)

$S(X_1X_2) :- NP(X_1), VP(X_2).$

$VP(\lambda x.X_2(\lambda y.X_1yx)) :- V(X_1), NP(X_2).$

$V(\lambda yx.X_2(X_1yx)(X_3yx)) :- V(X_1), Conj(X_2), V(X_3).$

$NP(X_1X_2) :- Det(X_1), N(X_2).$

$NP(\lambda u.u \mathbf{John}^e).$

$V(\mathbf{find}^{e \rightarrow e \rightarrow t}).$

$V(\mathbf{catch}^{e \rightarrow e \rightarrow t}).$

$Conj(\mathbf{\Lambda}^{t \rightarrow t \rightarrow t}).$

$Det(\lambda uv.\exists^{(e \rightarrow t) \rightarrow t}(\lambda y.\mathbf{\Lambda}^{t \rightarrow t \rightarrow t}(uy)(vy))).$

$N(\mathbf{unicorn}^{e \rightarrow t}).$

# Context-free $\lambda$ -term grammars (CFLGs)

$S(X_1X_2) :- NP(X_1), VP(X_2).$

$VP(\lambda x.X_2(\lambda y.X_1yx)) :- V(X_1), NP(X_2).$

$V(\lambda yx.X_2(X_1yx)(X_3yx)) :- V(X_1), Conj(X_2), V(X_3).$

$NP(X_1X_2) :- Det(X_1), N(X_2).$

$NP(\lambda u.u \mathbf{John}^e).$

$V(\mathbf{find}^{e \rightarrow e \rightarrow t}).$

$V(\mathbf{catch}^{e \rightarrow e \rightarrow t}).$

$Conj(\mathbf{\Lambda}^{t \rightarrow t \rightarrow t}).$

$Det(\lambda uv.\exists^{(e \rightarrow t) \rightarrow t}(\lambda y.\mathbf{\Lambda}^{t \rightarrow t \rightarrow t}(uy)(vy))).$

$N(\mathbf{unicorn}^{e \rightarrow t}).$

- $L(G) = \{ |M|_\beta \mid \vdash_G S(M) \}.$

# Context-free $\lambda$ -term grammars (CFLGs)

$$\begin{aligned} S(X_1 X_2) &:- NP(X_1), VP(X_2). \\ VP(\lambda x. X_2(\lambda y. X_1 y x)) &:- V(X_1), NP(X_2). \\ V(\lambda y x. X_2(X_1 y x)(X_3 y x)) &:- V(X_1), Conj(X_2), V(X_3). \\ NP(X_1 X_2) &:- Det(X_1), N(X_2). \\ NP(\lambda u. u \mathbf{John}^e) &. \\ V(\mathbf{find}^{e \rightarrow e \rightarrow t}) &. \\ V(\mathbf{catch}^{e \rightarrow e \rightarrow t}) &. \\ Conj(\mathbf{\wedge}^{t \rightarrow t \rightarrow t}) &. \\ Det(\lambda u v. \exists^{(e \rightarrow t) \rightarrow t}(\lambda y. \mathbf{\wedge}^{t \rightarrow t \rightarrow t}(u y)(v y))) &. \\ N(\mathbf{unicorn}^{e \rightarrow t}) &. \end{aligned}$$

- $L(G) = \{ |M|_\beta \mid \vdash_G S(M) \}$ .
- An alternative notation for **second-order abstract categorial grammars** (de Groote 2001), with the restriction to **linear**  $\lambda$ -terms dropped.
- With linear CFLGs, parsing/recognition complexity is in **P** (Salvati 2005).

## Representing string grammars with CFLGs

- Linear CFLGs can represent well-known mildly context-sensitive grammar formalisms like TAGs, MCTAGs, and MCFGs through encoding of strings as  $\lambda$ -terms (de Groote 2002, de Groote and Pogodalla 2004):

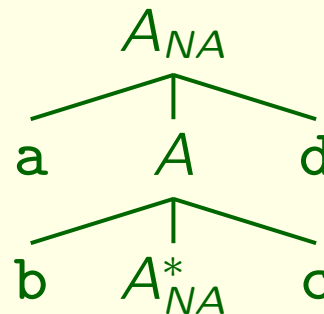
$$/a_1 \dots a_n/ = \lambda z. a_1^{o \rightarrow o} (\dots (a_n^{o \rightarrow o} z) \dots).$$

# Representing string grammars with CFLGs

- Linear CFLGs can represent well-known mildly context-sensitive grammar formalisms like TAGs, MCTAGs, and MCFGs through encoding of strings as  $\lambda$ -terms (de Groote 2002, de Groote and Pogodalla 2004):

$$/a_1 \dots a_n/ = \lambda z. a_1^{o \rightarrow o} (\dots (a_n^{o \rightarrow o} z) \dots).$$

S  
|  
A  
|  
ε



$S(\lambda y. X_1(\lambda z. z)y) :- A(X_1).$

$A(\lambda x y. a^{o \rightarrow o}(X_1(\lambda z. b^{o \rightarrow o}(x(c^{o \rightarrow o} z)))(d^{o \rightarrow o} y)))) :- A(X_1).$

$A(\lambda x y. xy).$

# Almost linear CFLGs

A  $\lambda$ -term  $M$  is **almost linear** if

- each occurrence of  $\lambda$  binds at least one occurrence of a variable ( $M$  is a  $\lambda$ -term), and
- any variable that occurs more than once in  $M$  is of atomic type.

$S(X_1X_2) :- NP(X_1), VP(X_2).$

$VP(\lambda x.X_2(\lambda y.X_1yx)) :- V(X_1), NP(X_2).$

$V(\lambda yx.X_2(X_1yx)(X_3yx)) :- V(X_1), Conj(X_2), V(X_3).$

$NP(X_1X_2) :- Det(X_1), N(X_2).$

$NP(\lambda u.u \mathbf{John}^e).$

$V(\mathbf{find}^{e \rightarrow e \rightarrow t}).$

$V(\mathbf{catch}^{e \rightarrow e \rightarrow t}).$

$Conj(\mathbf{\Lambda}^{t \rightarrow t \rightarrow t}).$

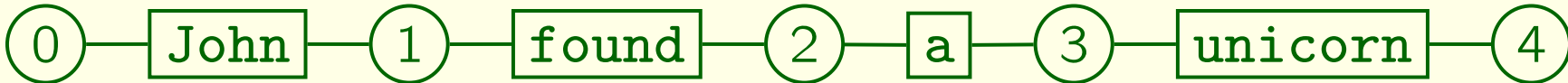
$Det(\lambda uv.\exists^{(e \rightarrow t) \rightarrow t}(\lambda y.\mathbf{\Lambda}^{t \rightarrow t \rightarrow t}(uy)(vy))).$

$N(\mathbf{unicorn}^{e \rightarrow t}).$

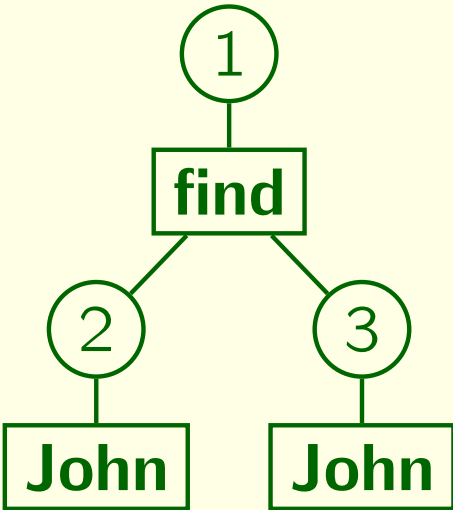
## Datalog representation of almost linear CFLGs

- input  $\lambda$ -term  $\Rightarrow$  (database, query)
- CFLG rule  $\Rightarrow$  Datalog rule

# Hypergraph representations of strings and trees



John found a unicorn

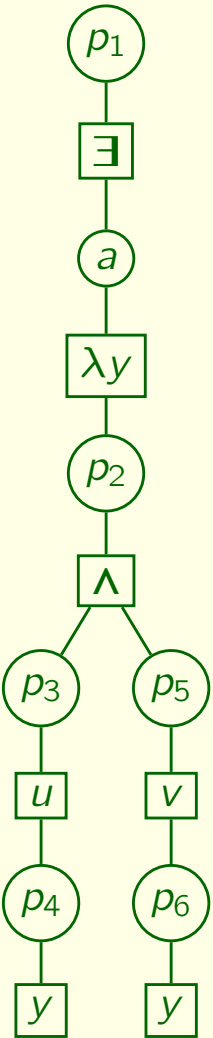


find John John

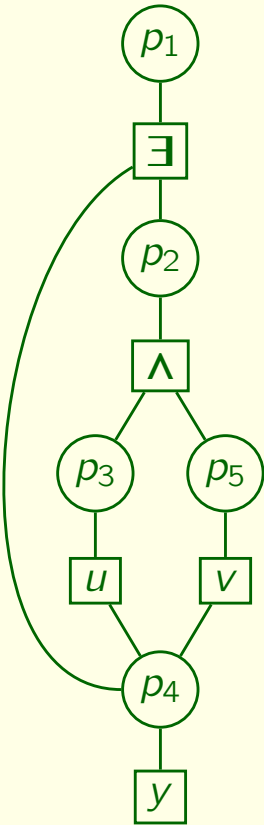
# Hypergraph representations of $\lambda$ -terms

$$\exists(\lambda y.\lambda(uy)(uy))$$

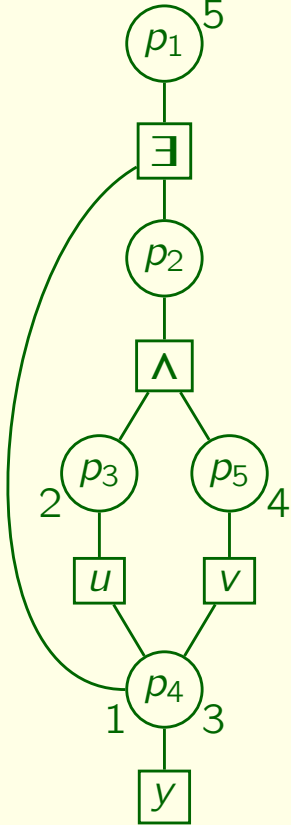
$$\lambda uv.\exists(\lambda y.\lambda(uy)(uy))$$



*tree graph*

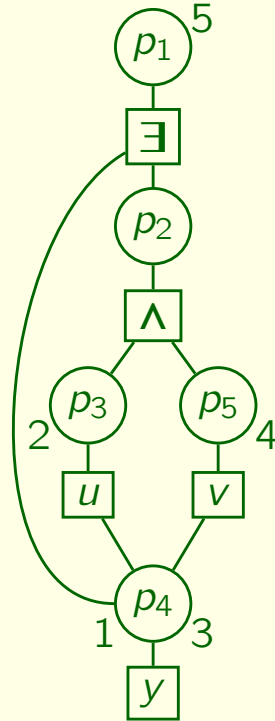


*term graph*



*term graph  
with external nodes*

# Meanings of graphs

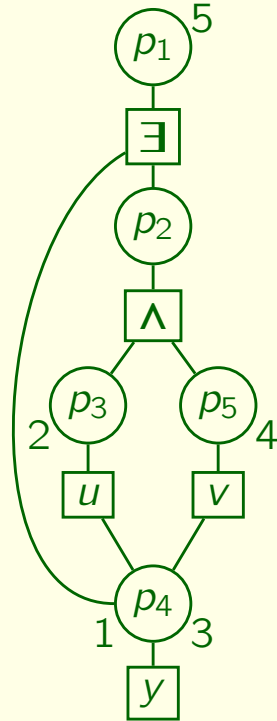


- The graph represents a **principal** (i.e., most general) **typing** of the term:

$$\exists : (e \rightarrow t) \rightarrow t, \Lambda : t \rightarrow t \rightarrow t$$

$$\vdash \lambda uv. \exists (\lambda y. \Lambda (uy) (vy)) : (e \rightarrow t) \rightarrow (e \rightarrow t) \rightarrow t$$

# Meanings of graphs

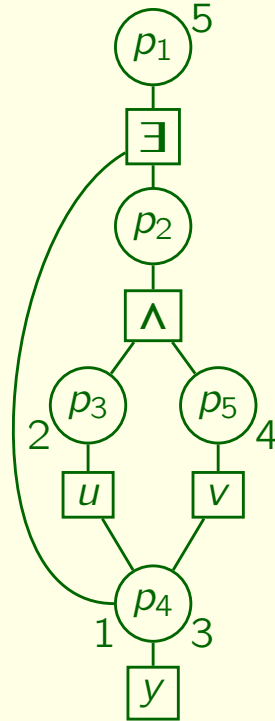


- The graph represents a **principal** (i.e., most general) **typing** of the term:

$$\exists : (p_4 \rightarrow p_2) \rightarrow p_1, \Lambda : p_3 \rightarrow p_5 \rightarrow p_2$$

$$\vdash \lambda uv. \exists (\lambda y. \Lambda (uy)(vy)) : (p_4 \rightarrow p_3) \rightarrow (p_4 \rightarrow p_5) \rightarrow p_1$$

# Meanings of graphs



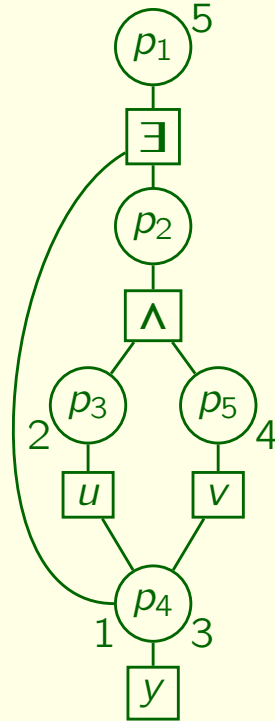
- The graph represents a **principal** (i.e., most general) **typing** of the term:

$$\exists : (\bar{p}_4 \rightarrow \bar{p}_2) \rightarrow \bar{p}_1, \wedge : \bar{p}_3 \rightarrow \bar{p}_5 \rightarrow \bar{p}_2$$

$$\vdash \lambda uv. \exists (\lambda y. \wedge (uy)(vy)) : (\bar{p}_4 \rightarrow \bar{p}_3) \rightarrow (\bar{p}_4 \rightarrow \bar{p}_5) \rightarrow \bar{p}_1$$

- If the  $\lambda$ -term is almost linear, the typing represented by the graph is **negatively non-duplicated** (cf. Aoto 1999).

# Meanings of graphs



- The graph represents a **principal** (i.e., most general) **typing** of the term:

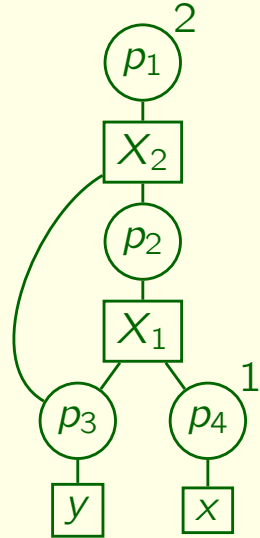
$$\exists : (\bar{p}_4 \rightarrow \bar{p}_2) \rightarrow \bar{p}_1, \Lambda : \bar{p}_3 \rightarrow \bar{p}_5 \rightarrow \bar{p}_2$$

$$\vdash \lambda uv. \exists (\lambda y. \Lambda (uy)(vy)) : (\bar{p}_4 \rightarrow \bar{p}_3) \rightarrow (\bar{p}_4 \rightarrow \bar{p}_5) \rightarrow \bar{p}_1$$

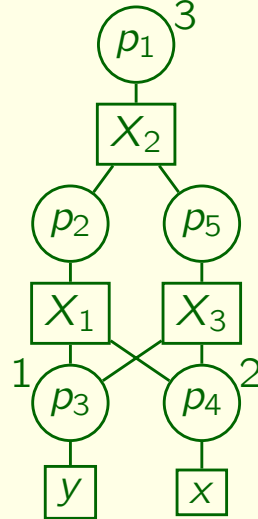
- If the  $\lambda$ -term is almost linear, the typing represented by the graph is **negatively non-duplicated** (cf. Aoto 1999).
- A negatively non-duplicated typing **uniquely determines** a  $\lambda$ -term up to  $\beta\eta$ -equality (Aoto and Ono 1994).

# Reduction to Datalog

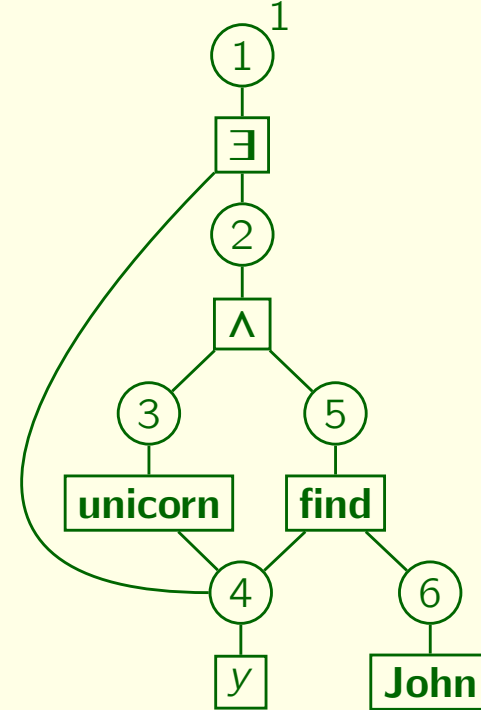
$\lambda x.X_2(\lambda y.X_1yx)$



$\lambda yx.X_2(X_1yx)(X_3yx)$



$\exists(\lambda y.\wedge(\text{unicorn } y)(\text{find } y \text{ John}))$



$S(p_1) :- NP(p_1, p_2, p_3), VP(p_2, p_3).$   
 $VP(p_1, p_4) :- V(p_2, p_4, p_3), NP(p_1, p_2, p_3).$   
 $V(p_1, p_4, p_3) :- V(p_2, p_4, p_3), Conj(p_1, p_5, p_2), V(p_5, p_4, p_3).$   
 $NP(p_1, p_4, p_5) :- Det(p_1, p_4, p_5, p_2, p_3), N(p_2, p_3).$   
 $NP(p_1, p_1, p_2) :- \text{John}(p_2).$   
 $V(p_1, p_3, p_2) :- \text{find}(p_1, p_3, p_2).$   
 $V(p_1, p_3, p_2) :- \text{catch}(p_1, p_3, p_2).$   
 $Conj(p_1, p_3, p_2) :- \wedge(p_1, p_3, p_2).$   
 $Det(p_1, p_5, p_4, p_3, p_4) :- \exists(p_1, p_2, p_4), \wedge(p_2, p_5, p_3).$   
 $N(p_1, p_2) :- \text{unicorn}(p_1, p_2).$

*Datalog program*

$\exists(1, 2, 4).$   
 $\wedge(2, 5, 3).$   
 $\text{unicorn}(3, 4).$   
 $\text{find}(5, 6, 4).$   
 $\text{John}(6).$

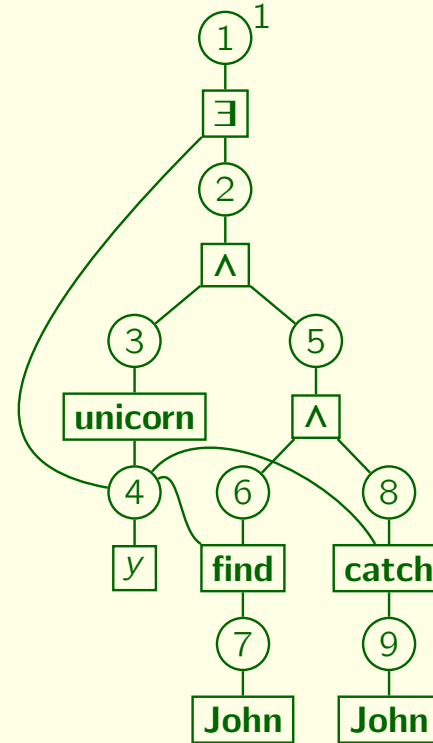
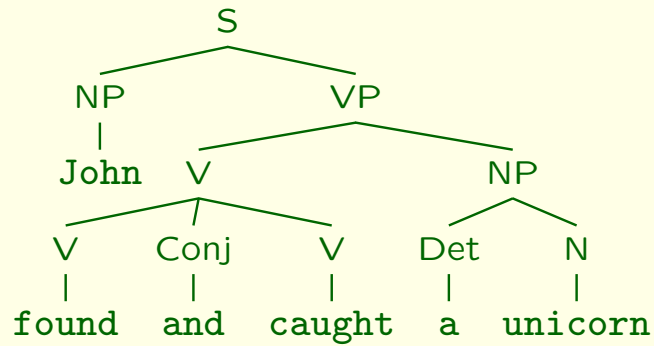
*database*

$? - S(1).$

*query*

# Reduction to Datalog

$\exists(\lambda y. \Lambda(\text{unicorn } y)(\Lambda(\text{find } y \text{ John})(\text{catch } y \text{ John}))))$



$S(p_1) :- NP(p_1, p_2, p_3), VP(p_2, p_3).$   
 $VP(p_1, p_4) :- V(p_2, p_4, p_3), NP(p_1, p_2, p_3).$   
 $V(p_1, p_4, p_3) :- V(p_2, p_4, p_3), Conj(p_1, p_5, p_2), V(p_5, p_4, p_3).$   
 $NP(p_1, p_4, p_5) :- Det(p_1, p_4, p_5, p_2, p_3), N(p_2, p_3).$   
 $NP(p_1, p_1, p_2) :- \text{John}(p_2).$   
 $V(p_1, p_3, p_2) :- \text{find}(p_1, p_3, p_2).$   
 $V(p_1, p_3, p_2) :- \text{catch}(p_1, p_3, p_2).$   
 $Conj(p_1, p_3, p_2) :- \Lambda(p_1, p_3, p_2).$   
 $Det(p_1, p_5, p_4, p_3, p_4) :- \exists(p_1, p_2, p_4), \Lambda(p_2, p_5, p_3).$   
 $N(p_1, p_2) :- \text{unicorn}(p_1, p_2).$

*Datalog program*

$\exists(1, 2, 4).$   
 $\Lambda(2, 5, 3).$   
 $\text{unicorn}(3, 4).$   
 $\Lambda(5, 8, 6).$   
 $\text{find}(6, 7, 4).$   
 $\text{John}(7).$   
 $\text{catch}(8, 9, 4).$   
 $\text{John}(9).$

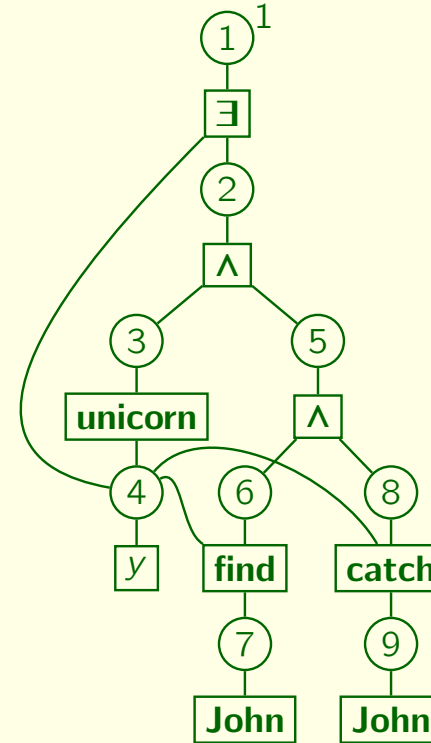
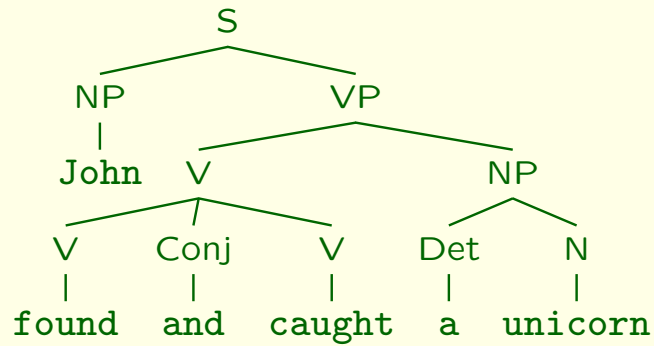
*database*

$?- S(1).$

*query*

# Reduction to Datalog

$\exists(\lambda y. \Lambda(\text{unicorn } y)(\Lambda(\text{find } y \text{ John})(\text{catch } y \text{ John}))))$



$S(p_1) :- NP(p_1, p_2, p_3), VP(p_2, p_3).$   
 $VP(p_1, p_4) :- V(p_2, p_4, p_3), NP(p_1, p_2, p_3).$   
 $V(p_1, p_4, p_3) :- V(p_2, p_4, p_3), Conj(p_1, p_5, p_2), V(p_5, p_4, p_3).$   
 $NP(p_1, p_4, p_5) :- Det(p_1, p_4, p_5, p_2, p_3), N(p_2, p_3).$   
 $NP(p_1, p_1, p_2) :- \text{John}(p_2).$   
 $V(p_1, p_3, p_2) :- \text{find}(p_1, p_3, p_2).$   
 $V(p_1, p_3, p_2) :- \text{catch}(p_1, p_3, p_2).$   
 $Conj(p_1, p_3, p_2) :- \Lambda(p_1, p_3, p_2).$   
 $Det(p_1, p_5, p_4, p_3, p_4) :- \exists(p_1, p_2, p_4), \Lambda(p_2, p_5, p_3).$   
 $N(p_1, p_2) :- \text{unicorn}(p_1, p_2).$

*Datalog program*

$\exists(1, 2, 4).$   
 $\Lambda(2, 5, 3).$   
 $\text{unicorn}(3, 4).$   
 $\Lambda(5, 8, 6).$   
 $\text{find}(6, 7, 4).$   
 $\text{John}(7).$   
 $\text{catch}(8, 9, 4).$   
 $\text{John}(9).$

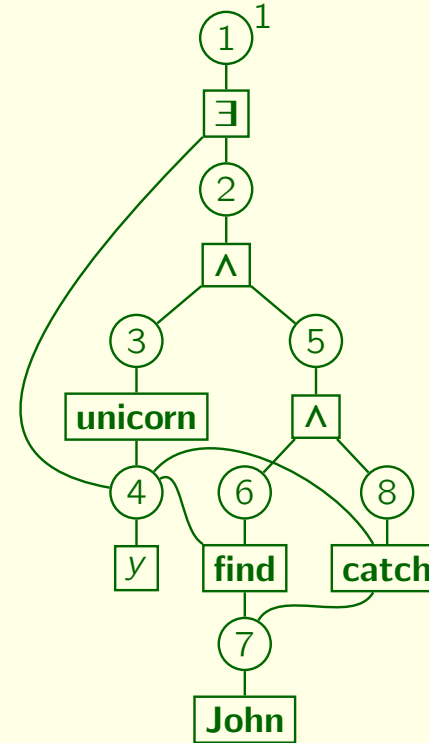
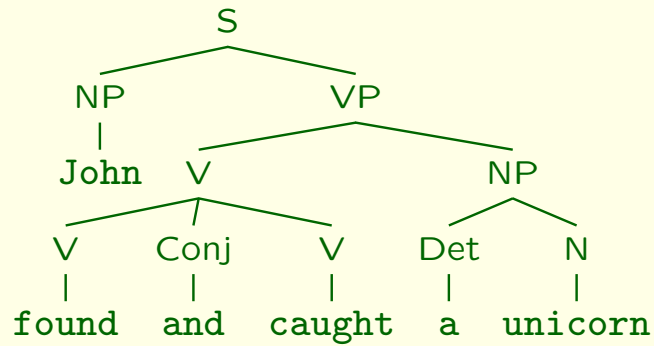
*database*

$?- S(1).$   
*query*

No!

# Reduction to Datalog

$\exists(\lambda y. \Lambda(\text{unicorn } y)(\Lambda(\text{find } y \text{ John})(\text{catch } y \text{ John}))))$



$S(p_1) :- NP(p_1, p_2, p_3), VP(p_2, p_3).$   
 $VP(p_1, p_4) :- V(p_2, p_4, p_3), NP(p_1, p_2, p_3).$   
 $V(p_1, p_4, p_3) :- V(p_2, p_4, p_3), Conj(p_1, p_5, p_2), V(p_5, p_4, p_3).$   
 $NP(p_1, p_4, p_5) :- Det(p_1, p_4, p_5, p_2, p_3), N(p_2, p_3).$   
 $NP(p_1, p_1, p_2) :- \text{John}(p_2).$   
 $V(p_1, p_3, p_2) :- \text{find}(p_1, p_3, p_2).$   
 $V(p_1, p_3, p_2) :- \text{catch}(p_1, p_3, p_2).$   
 $Conj(p_1, p_3, p_2) :- \Lambda(p_1, p_3, p_2).$   
 $Det(p_1, p_5, p_4, p_3, p_4) :- \exists(p_1, p_2, p_4), \Lambda(p_2, p_5, p_3).$   
 $N(p_1, p_2) :- \text{unicorn}(p_1, p_2).$

*Datalog program*

$\exists(1, 2, 4).$   
 $\Lambda(2, 5, 3).$   
 $\text{unicorn}(3, 4).$   
 $\Lambda(5, 8, 6).$   
 $\text{find}(6, 7, 4).$   
 $\text{John}(7).$   
 $\text{catch}(8, 7, 4).$

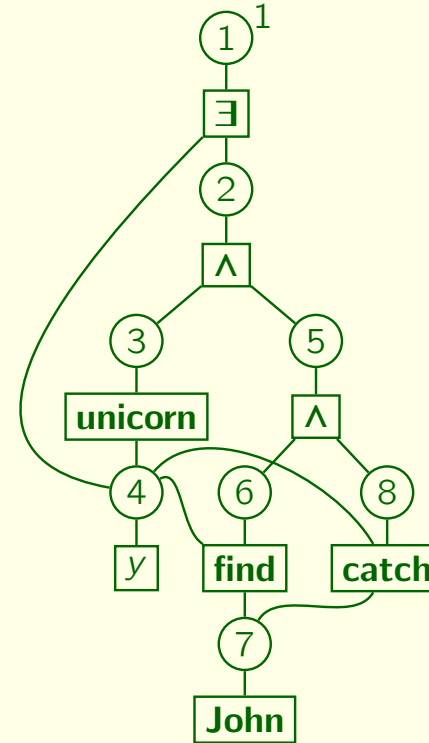
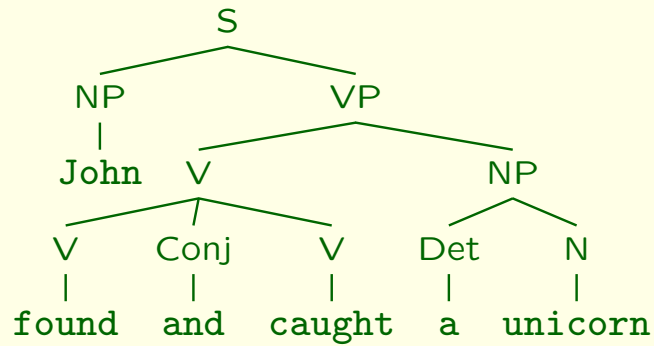
*database*

$? - S(1).$

*query*

# Reduction to Datalog

$\exists(\lambda y. \Lambda(\text{unicorn } y)(\Lambda(\text{find } y \text{ John})(\text{catch } y \text{ John}))))$



$S(p_1) :- NP(p_1, p_2, p_3), VP(p_2, p_3).$   
 $VP(p_1, p_4) :- V(p_2, p_4, p_3), NP(p_1, p_2, p_3).$   
 $V(p_1, p_4, p_3) :- V(p_2, p_4, p_3), Conj(p_1, p_5, p_2), V(p_5, p_4, p_3).$   
 $NP(p_1, p_4, p_5) :- Det(p_1, p_4, p_5, p_2, p_3), N(p_2, p_3).$   
 $NP(p_1, p_1, p_2) :- \text{John}(p_2).$   
 $V(p_1, p_3, p_2) :- \text{find}(p_1, p_3, p_2).$   
 $V(p_1, p_3, p_2) :- \text{catch}(p_1, p_3, p_2).$   
 $Conj(p_1, p_3, p_2) :- \Lambda(p_1, p_3, p_2).$   
 $Det(p_1, p_5, p_4, p_3, p_4) :- \exists(p_1, p_2, p_4), \Lambda(p_2, p_5, p_3).$   
 $N(p_1, p_2) :- \text{unicorn}(p_1, p_2).$

*Datalog program*

$\exists(1, 2, 4).$   
 $\Lambda(2, 5, 3).$   
 $\text{unicorn}(3, 4).$   
 $\Lambda(5, 8, 6).$   
 $\text{find}(6, 7, 4).$   
 $\text{John}(7).$   
 $\text{catch}(8, 7, 4).$

*database*

$? - S(1).$

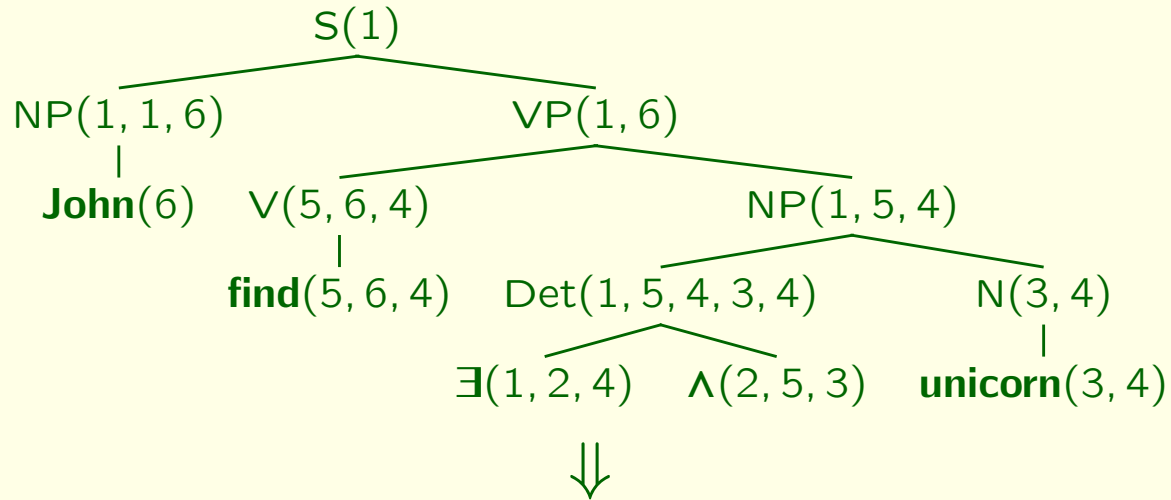
*query*

Yes!

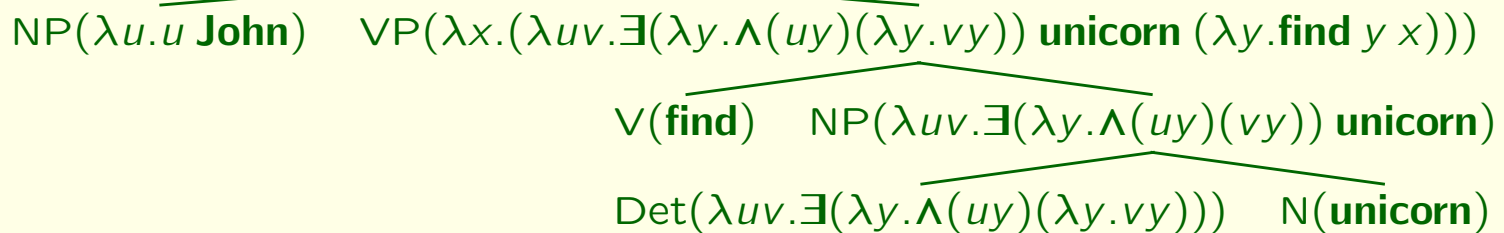
## Reduction to Datalog

- Given an input  $\lambda$ -term, find the **most compact term graph** representing it.

# From Datalog derivation to grammar derivation



$S((\lambda u.u \text{ John})(\lambda x.(\lambda uv.\exists(\lambda y.\wedge(uy)(vy)) \text{ unicorn } (\lambda y.\text{find } y x))))$



$\text{John} : 6, \text{find} : 4 \rightarrow 6 \rightarrow 5, \exists : (4 \rightarrow 2) \rightarrow 1, \wedge : 3 \rightarrow 5 \rightarrow 2, \text{unicorn} : 4 \rightarrow 3$

$\vdash (\lambda u.u \text{ John})(\lambda x.(\lambda uv.\exists(\lambda y.\wedge(uy)(vy)) \text{ unicorn } (\lambda y.\text{find } y x))) : 1$

*Aoto and Ono's Coherence Theorem*  $\Downarrow$

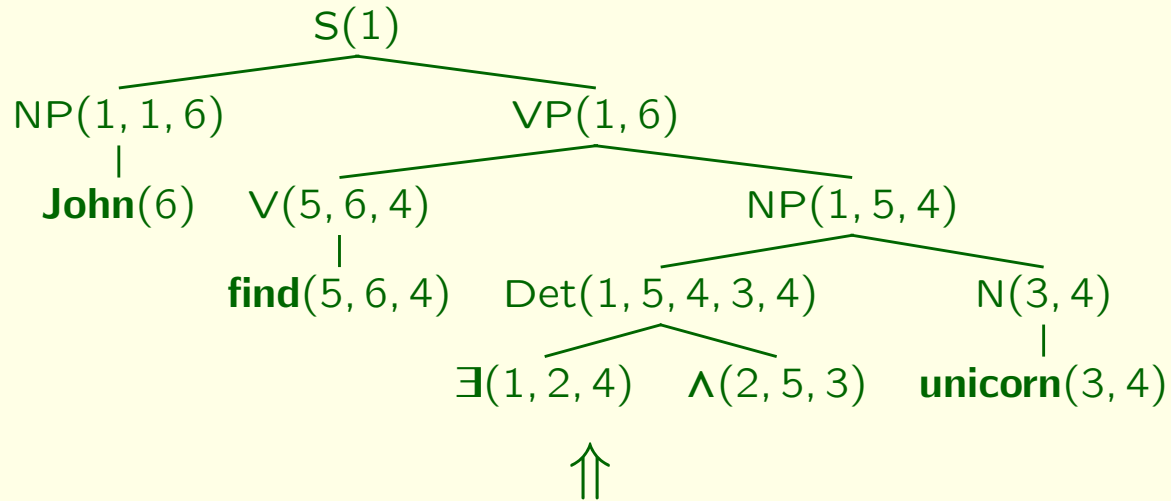
$(\lambda u.u \text{ John})(\lambda x.(\lambda uv.\exists(\lambda y.\wedge(uy)(vy)) \text{ unicorn } (\lambda y.\text{find } y x)))$

$\rightarrow_{\beta} \exists(\lambda y.\wedge(\text{unicorn } x)(\text{find } y \text{ John}))$

$\Downarrow$

$\exists(\lambda y.\wedge(\text{unicorn } y)(\text{find } y \text{ John})) \in L(G)$

# From grammar derivation to Datalog derivation

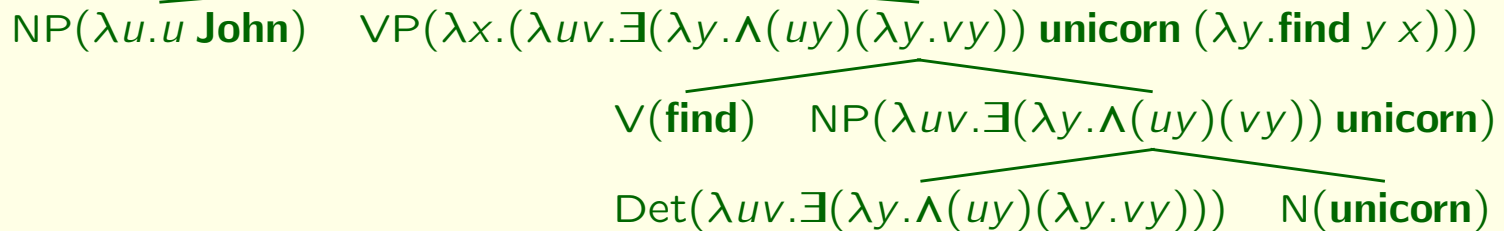


**John** : 6, **find** : 4 → 6 → 5, **∃** : (4 → 2) → 1, **∧** : 3 → 5 → 2, **unicorn** : 4 → 3

$\vdash (\lambda u.u \text{ John})(\lambda x.(\lambda uv.\exists(\lambda y.\wedge(uy)(vy)) \text{ unicorn } (\lambda y.\text{find } y x)))) : 1$

*Subject Expansion*  $\Uparrow$

$S((\lambda u.u \text{ John})(\lambda x.(\lambda uv.\exists(\lambda y.\wedge(uy)(vy)) \text{ unicorn } (\lambda y.\text{find } y x))))$



$(\lambda u.u \text{ John})(\lambda x.(\lambda uv.\exists(\lambda y.\wedge(uy)(vy)) \text{ unicorn } (\lambda y.\text{find } y x)))$

$\rightarrow_{\beta} \exists(\lambda y.\wedge(\text{unicorn } x)(\text{find } y \text{ John}))$

$\Uparrow$

$\exists(\lambda y.\wedge(\text{unicorn } x)(\text{find } y \text{ John})) \in L(G)$

# Tight complexity bound

- “ $\epsilon$ -rule”:

$$B(M)$$

where  $M$  is a combinator.

- $\epsilon$ -rules can be eliminated from almost linear CFLGs (cf. Kanazawa and Yoshinaka 2005).

## Tight complexity bound

- “ $\epsilon$ -rule”:

$$B(M)$$

where  $M$  is a **combinator**.

- $\epsilon$ -rules can be eliminated from almost linear CFLGs (cf. Kanazawa and Yoshinaka 2005).
- If  $G$  is an almost linear CFLG without  $\epsilon$ -rules and  $M$  is the normal form of an almost linear  $\lambda$ -term, any Datalog derivation tree for

$$\text{program}(G) \cup \text{database}(M) \vdash \text{query}(M)$$

has a number of leaves linear in the size of  $M$ .

## Tight complexity bound

- “ $\epsilon$ -rule”:

$$B(M)$$

where  $M$  is a **combinator**.

- $\epsilon$ -rules can be eliminated from almost linear CFLGs (cf. Kanazawa and Yoshinaka 2005).
- If  $G$  is an almost linear CFLG without  $\epsilon$ -rules and  $M$  is the normal form of an almost linear  $\lambda$ -term, any Datalog derivation tree for

$$\text{program}(G) \cup \text{database}(M) \vdash \text{query}(M)$$

has a number of leaves linear in the size of  $M$ .

- $(\text{database}(M), \text{query}(M))$  can be computed in **logspace**.

# Tight complexity bound

- “ $\epsilon$ -rule”:

$$B(M)$$

where  $M$  is a **combinator**.

- $\epsilon$ -rules can be eliminated from almost linear CFLGs (cf. Kanazawa and Yoshinaka 2005).
- If  $G$  is an almost linear CFLG without  $\epsilon$ -rules and  $M$  is the normal form of an almost linear  $\lambda$ -term, any Datalog derivation tree for

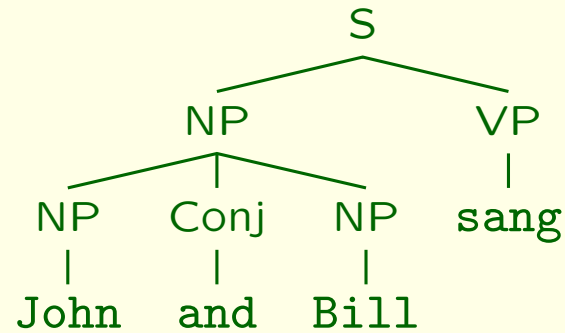
$$\text{program}(G) \cup \text{database}(M) \vdash \text{query}(M)$$

has a number of leaves linear in the size of  $M$ .

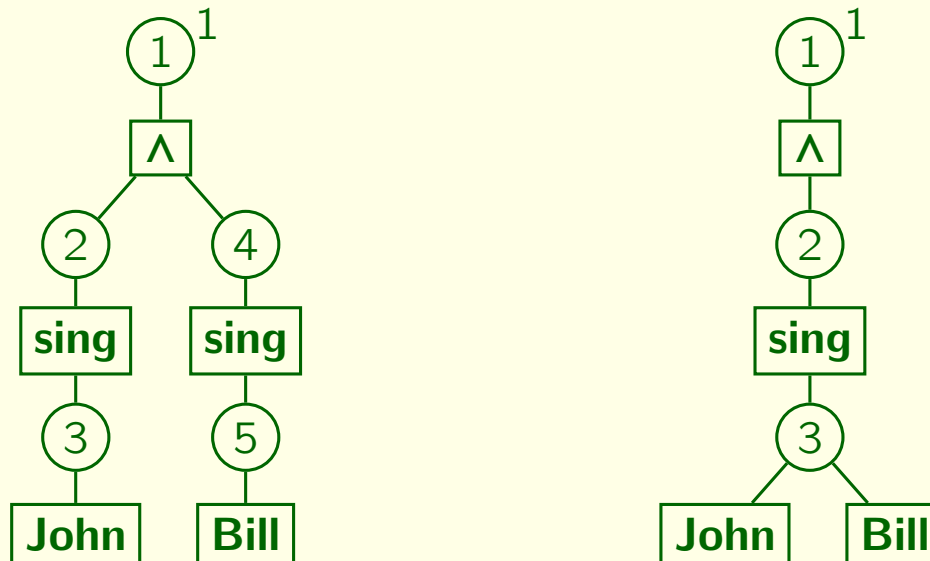
- $(\text{database}(M), \text{query}(M))$  can be computed in **logspace**.
- This implies that  $L(G)$  belongs to **LOGCFL** (cf. Ullman and Van Gelder 1988, Kanellakis 1988).

# Limitation

$NP(\lambda x^{e \rightarrow t}. X_2(X_1 x)(X_3 x)) :- NP(X_1), Conj(X_2), NP(X_3).$



$\wedge(\text{sing John})(\text{sing Bill})$



*not a term graph*

## Conclusion

- Parsing and surface realization are reduced to Datalog query evaluation, which can be computed in polynomial time in the size of the database.
- The reduction provides a more convincingly **uniform architecture** for parsing and generation than previous approaches.
- The reduction is to the **LOGCFL** fragment of Datalog, which implies the existence of fast **parallel** algorithms.
- Sophisticated evaluation techniques for Datalog can be applied to parsing and generation. In particular, **generalized supplementary magic-sets** rewriting (Beeri and Ramakrishnan 1991) automatically yields **Earley-style** algorithms for both parsing and generation.
- Higher-order  $\lambda$ -terms need not be avoided for the purpose of achieving computational efficiency.