Parsing and Generation as Datalog Queries

Makoto Kanazawa

National Institute of Informatics

CFG recognition/parsing as Datalog query evaluation

(0) — **John** — (1) — **found** — (2) — **a** — (3) — **unicorn** — (4)

$\mathsf{S}(i,j) :- \mathsf{NP}(i,k), \mathsf{VP}(k,j).$

$\mathsf{VP}(i,j) :- \mathsf{V}(i,k), \mathsf{NP}(k,j).$

$\mathsf{NP}(i,j) :- \mathsf{Det}(i,k), \mathsf{N}(k,j).$

$\mathsf{NP}(i,j) :- \mathtt{John}(i,j).$

$\mathsf{V}(i,j) :- \mathtt{found}(i,j).$

$\mathsf{Det}(i,j) :- \mathtt{a}(i,j).$

$\mathsf{N}(i,j) :- \mathtt{unicorn}(i,j).$

Datalog program

$\mathtt{John}(0,1).$

$\mathtt{found}(1,2).$

$\mathtt{a}(2,3).$

$\mathtt{unicorn}(3,4).$

database

$?- \mathsf{S}(0,4).$

query

CFG recognition/parsing as Datalog query evaluation

$0 - \texttt{John} - 1 - \texttt{found} - 2 - \texttt{a} - 3 - \texttt{unicorn} - 4$

$$
\begin{aligned}
&\mathrm{S}(i,j) :- \mathrm{NP}(i,k), \mathrm{VP}(k,j).\\
&\mathrm{VP}(i,j) :- \mathrm{V}(i,k), \mathrm{NP}(k,j).\\
&\mathrm{NP}(i,j) :- \mathrm{Det}(i,k), \mathrm{N}(k,j).\\
&\mathrm{NP}(i,j) :- \texttt{John}(i,j).\\
&\mathrm{V}(i,j) :- \texttt{found}(i,j).\\
&\mathrm{Det}(i,j) :- \texttt{a}(i,j).\\
&\mathrm{N}(i,j) :- \texttt{unicorn}(i,j).
\end{aligned}
$$

Datalog program

$$
\begin{aligned}
&\texttt{John}(0,1).\\
&\texttt{found}(1,2).\\
&\texttt{a}(2,3).\\
&\texttt{unicorn}(3,4).
\end{aligned}
$$

database

$?- \mathrm{S}(0,4).$

query

- Polynomial-time algorithms
  - CYK ≈ seminaive bottom-up evaluation
  - Earley ≈ magic-sets rewriting

Extending Datalog representation

CFG recognition/parsing

recognition/parsing for	surface realization for
grammars with	CFGs with
"context-free" derivations	Montague semantics
(MC)TAG	(suitably restricted)
(P)MCFG	

Extending Datalog representation

CFG recognition/parsing

recognition/parsing for	surface realization for
grammars with	CFGs with
“context-free” derivations	Montague semantics
(MC)TAG	(suitably restricted)
(P)MCFG	

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

Extending Datalog representation

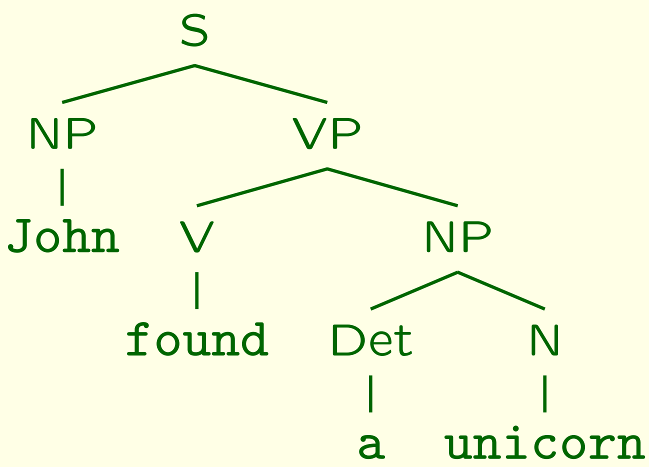
CFG recognition/parsing

recognition/parsing for	surface realization for
grammars with	CFGs with
"context-free" derivations	Montague semantics
(MC)TAG	(suitably restricted)
(P)MCFG	

- 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



$$
\begin{aligned}
&\mathsf{S}(X_1 X_2) \to \mathsf{NP}(X_1)\ \ \mathsf{VP}(X_2)\\
&\mathsf{VP}(\lambda x.X_2(\lambda y.X_1 y x)) \to \mathsf{V}(X_1)\ \ \mathsf{NP}(X_2)\\
&\mathsf{V}(\lambda y x.X_2(X_1 y x)(X_3 y x)) \to \mathsf{V}(X_1)\ \ \mathsf{Conj}(X_2)\ \ \mathsf{V}(X_3)\\
&\mathsf{NP}(X_1 X_2) \to \mathsf{Det}(X_1)\ \ \mathsf{N}(X_2)\\
&\mathsf{NP}(\lambda u.u\,\mathbf{John}^{e}) \to \texttt{John}\\
&\mathsf{V}(\mathbf{find}^{e\to e\to t}) \to \texttt{found}\\
&\mathsf{V}(\mathbf{catch}^{e\to e\to t}) \to \texttt{caught}\\
&\mathsf{Conj}(\wedge^{t\to t\to t}) \to \texttt{and}\\
&\mathsf{Det}(\lambda u v.\exists^{(e\to t)\to t}(\lambda y.\wedge^{t\to t\to t}(u y)(v y))) \to \texttt{a}\\
&\mathsf{N}(\mathbf{unicorn}^{e\to t}) \to \texttt{unicorn}
\end{aligned}
$$

$$
\begin{aligned}
&(\lambda u.u\,\mathbf{John})(\lambda x.(\lambda u v.\exists(\lambda y.\wedge(u y)(v y)))\,\mathbf{unicorn}\,(\lambda y.\mathbf{find}\,y\,x))\\
&\twoheadrightarrow_\beta\ \exists(\lambda y.\wedge(\mathbf{unicorn}\,y)(\mathbf{find}\,y\,\mathbf{John}))\\
&\approx \exists y(\mathbf{unicorn}(y) \wedge \mathbf{find}(\mathbf{John}, y))
\end{aligned}
$$

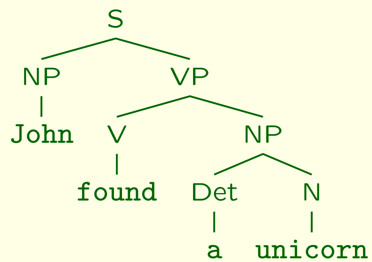
logical form

Semantic interpretation and surface realization

Semantic interpretation

string — *derivation tree* — *logical form*

`John found a unicorn` $\Longrightarrow$ $O(n^3)$

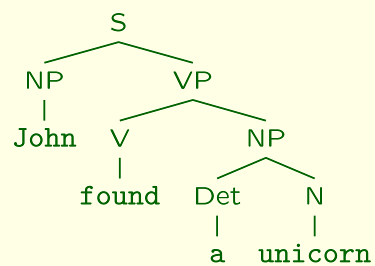


$\Longrightarrow$ easy $\exists(\lambda y.\wedge(\textbf{unicorn}\ y)(\textbf{find}\ y\ \textbf{John}))$

Surface realization (tactical generation)

logical form — *derivation tree* — *string*

$\exists(\lambda y.\wedge(\textbf{unicorn}\ y)(\textbf{find}\ y\ \textbf{John}))$ $\Longrightarrow$?



$\Longrightarrow$ easy `John found a unicorn`

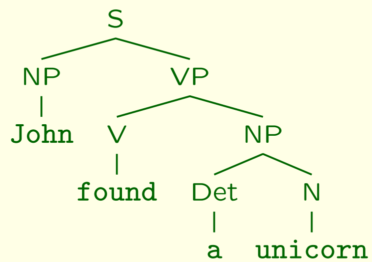
Semantic interpretation and surface realization

Parsing (of input string)

string *derivation tree*

`John found a unicorn` $\Longrightarrow$

$O(n^3)$

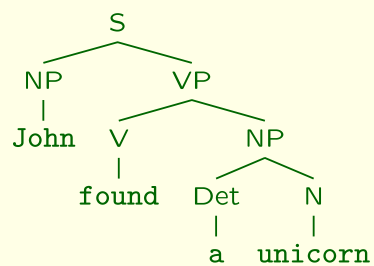


Parsing of input logical form

logical form *derivation tree*

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

?



Semantic interpretation and surface realization

Recognition (of input string)

string *grammatical?*

`John found a unicorn` $\Longrightarrow$ yes/no

$O(n^3)$

Recognition of input logical form

logical form *surface realizable?*

$\exists(\lambda y.\wedge(\textbf{unicorn}\ y)(\textbf{find}\ y\ \textbf{John}))$ $\Longrightarrow$ 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 λ-terms, and the computation of meaning involves β-reduction.
- More faithful to standard linguistic analysis than previous approaches.

Synchronous grammar view

$$
\begin{array}{l}
\mathsf{S}(X_1 X_2) \rightarrow \mathsf{NP}(X_1)\ \ \mathsf{VP}(X_2) \\
\mathsf{VP}(\lambda x.X_2(\lambda y.X_1 y x)) \rightarrow \mathsf{V}(X_1)\ \ \mathsf{NP}(X_2) \\
\mathsf{V}(\lambda y x.X_2(X_1 y x)(X_3 y x)) \rightarrow \mathsf{V}(X_1)\ \ \mathsf{Conj}(X_2)\ \ \mathsf{V}(X_3) \\
\mathsf{NP}(X_1 X_2) \rightarrow \mathsf{Det}(X_1)\ \ \mathsf{N}(X_2) \\
\mathsf{NP}(\lambda u.u\ \mathbf{John}^e) \rightarrow \texttt{John} \\
\mathsf{V}(\mathbf{find}^{e\rightarrow e\rightarrow t}) \rightarrow \texttt{found} \\
\mathsf{V}(\mathbf{catch}^{e\rightarrow e\rightarrow t}) \rightarrow \texttt{caught} \\
\mathsf{Conj}(\wedge^{t\rightarrow t\rightarrow t}) \rightarrow \texttt{and} \\
\mathsf{Det}(\lambda u v.\exists^{(e\rightarrow t)\rightarrow t}(\lambda y.\wedge^{t\rightarrow t\rightarrow t}(u y)(v y))) \rightarrow \texttt{a} \\
\mathsf{N}(\mathbf{unicorn}^{e\rightarrow t}) \rightarrow \texttt{unicorn}
\end{array}
$$

string grammar

$$
\begin{array}{l}
\mathsf{S} \rightarrow \mathsf{NP}\ \ \mathsf{VP} \\
\mathsf{VP} \rightarrow \mathsf{V}\ \ \mathsf{NP} \\
\mathsf{V} \rightarrow \mathsf{V}\ \ \mathsf{Conj}\ \ \mathsf{V} \\
\mathsf{NP} \rightarrow \mathsf{Det}\ \ \mathsf{N} \\
\mathsf{NP} \rightarrow \texttt{John} \\
\mathsf{V} \rightarrow \texttt{found} \\
\mathsf{V} \rightarrow \texttt{caught} \\
\mathsf{Conj} \rightarrow \texttt{and} \\
\mathsf{Det} \rightarrow \texttt{a} \\
\mathsf{N} \rightarrow \texttt{unicorn}
\end{array}
$$

λ-term grammar

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

Context-free λ-term grammars (CFLGs)

$\mathsf{S}(X_1 X_2) \mathrel{:-} \mathsf{NP}(X_1), \mathsf{VP}(X_2).$

$\mathsf{VP}(\lambda x. X_2(\lambda y. X_1 y x)) \mathrel{:-} \mathsf{V}(X_1), \mathsf{NP}(X_2).$

$\mathsf{V}(\lambda y x. X_2(X_1 y x)(X_3 y x)) \mathrel{:-} \mathsf{V}(X_1), \mathsf{Conj}(X_2), \mathsf{V}(X_3).$

$\mathsf{NP}(X_1 X_2) \mathrel{:-} \mathsf{Det}(X_1), \mathsf{N}(X_2).$

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

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

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

$\mathsf{Conj}(\boldsymbol{\wedge}^{t \to t \to t}).$

$\mathsf{Det}(\lambda u v. \boldsymbol{\exists}^{(e \to t) \to t}(\lambda y. \boldsymbol{\wedge}^{t \to t \to t}(u y)(v y))).$

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

Context-free λ-term grammars (CFLGs)

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

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

Context-free λ-term grammars (CFLGs)

$\mathsf{S}(X_1X_2) :\!-\ \mathsf{NP}(X_1), \mathsf{VP}(X_2).$

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

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

$\mathsf{NP}(X_1X_2) :\!-\ \mathsf{Det}(X_1), \mathsf{N}(X_2).$

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

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

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

$\mathsf{Conj}(\wedge^{t\to t\to t}).$

$\mathsf{Det}(\lambda uv.\exists^{(e\to t)\to t}(\lambda y.\wedge^{t\to t\to t}(uy)(vy))).$

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

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

Representing string grammars with CFLGs

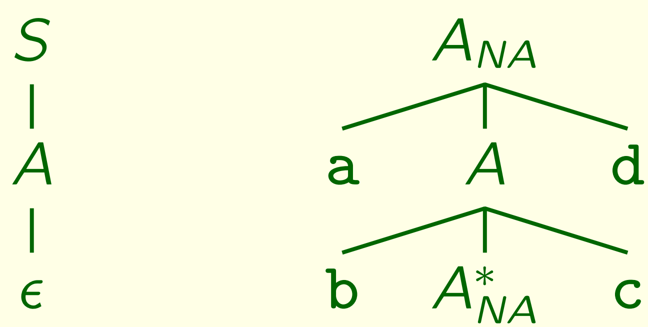
- Linear CFLGs can represent well-known mildly context-sensitive grammar formalisms like TAGs, MCTAGs, and MCFGs through encoding of strings as λ-terms (de Groote 2002, de Groote and Pogodalla 2004):

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

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 λ-terms (de Groote 2002, de Groote and Pogodalla 2004):

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



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

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

$A(\lambda xy.xy).$

Almost linear CFLGs

A λ-term M is almost linear if

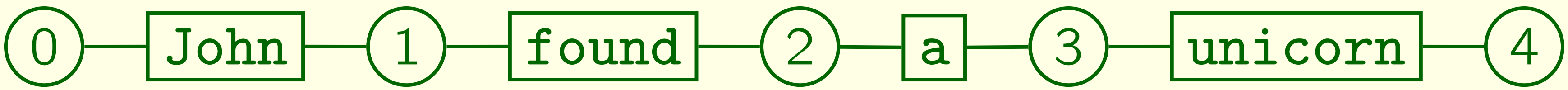
- each occurrence of λ binds at least one occurrence of a variable (M is a λI-term), and
- any variable that occurs more than once in M is of atomic type.

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

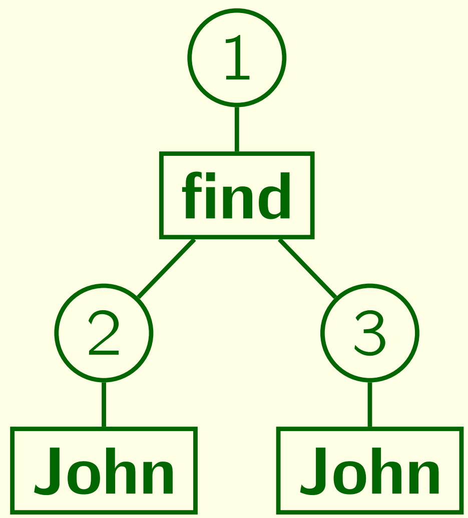
Datalog representation of almost linear CFLGs

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

Hypergraph representations of strings and trees



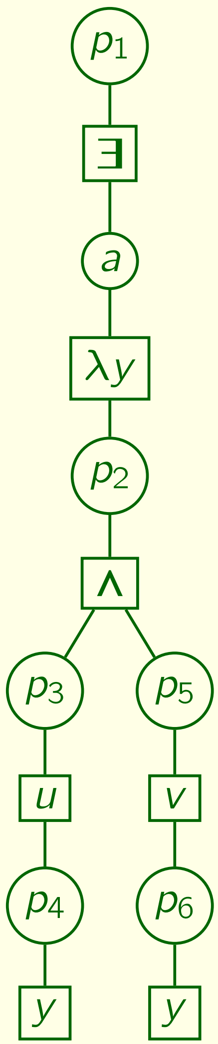
John found a unicorn



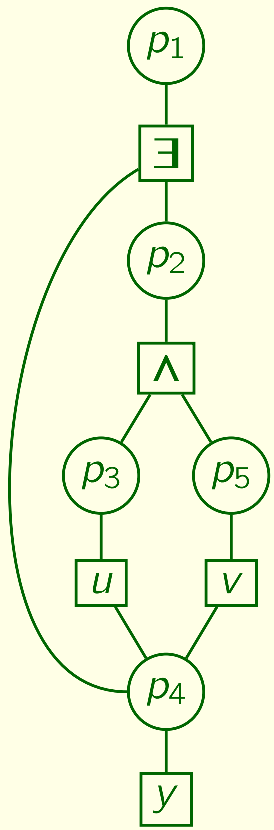
find John John

Hypergraph representations of λ-terms

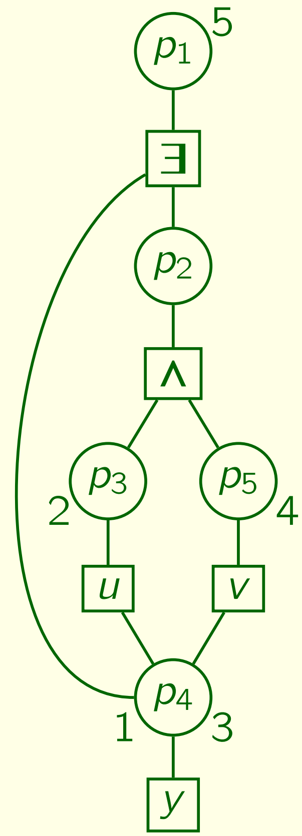
$\exists(\lambda y.\wedge(uy)(uy))$ $\lambda uv.\exists(\lambda y.\wedge(uy)(uy))$



tree graph

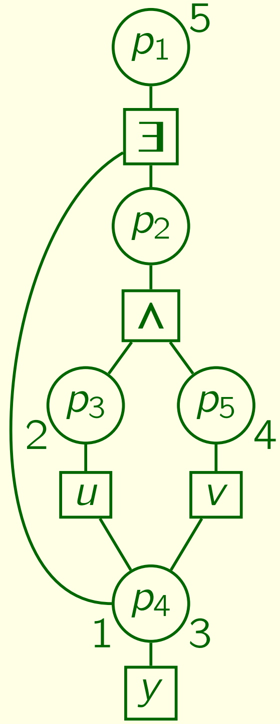


term graph



term graph
with external nodes

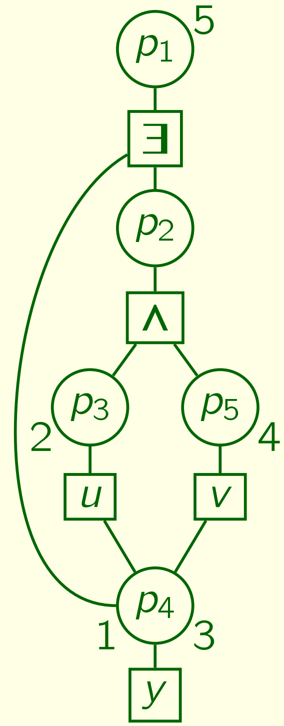
Meanings of graphs



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

$$\exists : (e \to t) \to t, \wedge : t \to t \to t$$
$$\vdash \lambda uv.\exists(\lambda y.\wedge(uy)(vy)) : (e \to t) \to (e \to t) \to t$$

Meanings of graphs

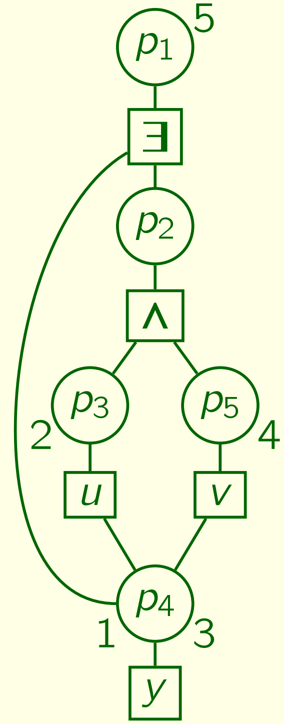


- The graph represents a principal (i.e., most general) typing of the term:

$$\exists : (p_4 \to p_2) \to p_1, \land : p_3 \to p_5 \to p_2$$

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

Meanings of graphs

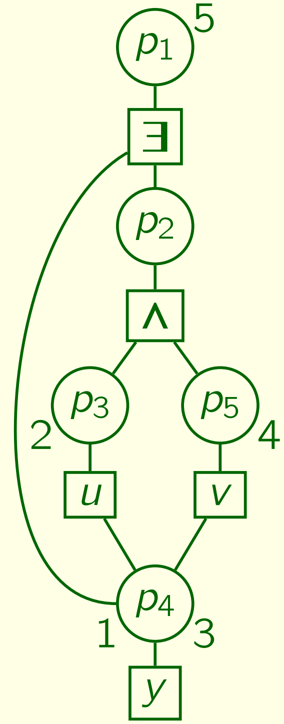


- The graph represents a principal (i.e., most general) typing of the term:

$$\exists : (\overset{-}{p}_4 \to \overset{+}{p}_2) \to \overset{-}{p}_1, \land : \overset{+}{p}_3 \to \overset{+}{p}_5 \to \overset{-}{p}_2$$
$$\vdash \lambda uv.\exists(\lambda y.\land(uy)(vy)) : (\overset{+}{p}_4 \to \overset{-}{p}_3) \to (\overset{+}{p}_4 \to \overset{-}{p}_5) \to \overset{+}{p}_1$$

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

Meanings of graphs



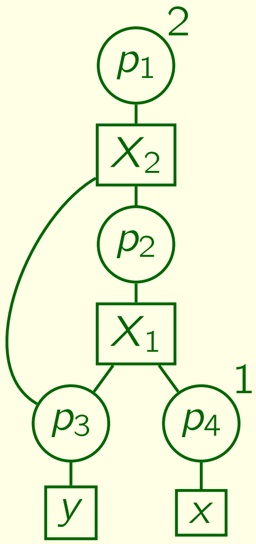
- The graph represents a principal (i.e., most general) typing of the term:

$$\exists : (\overset{-}{p_4} \to \overset{+}{p_2}) \to \overset{-}{p_1}, \wedge : \overset{+}{p_3} \to \overset{+}{p_5} \to \overset{-}{p_2}$$
$$\vdash \lambda uv.\exists(\lambda y.\wedge(uy)(vy)) : (\overset{+}{p_4} \to \overset{-}{p_3}) \to (\overset{+}{p_4} \to \overset{-}{p_5}) \to \overset{+}{p_1}$$

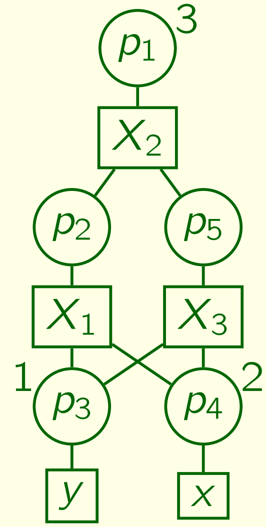
- If the λ-term is almost linear, the typing represented by the graph is negatively non-duplicated (cf. Aoto 1999).
- A negatively non-duplicated typing uniquely determines a λ-term up to $\beta\eta$-equality (Aoto and Ono 1994).

Reduction to Datalog

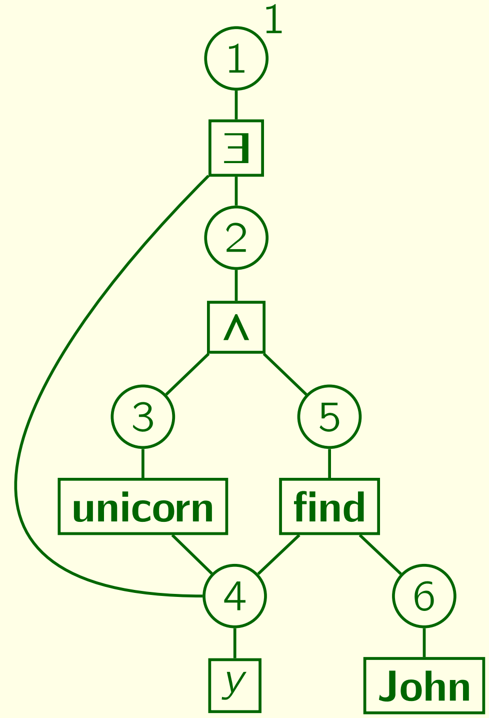
$\lambda x.X_2(\lambda y.X_1 yx)$



$\lambda yx.X_2(X_1 yx)(X_3 yx)$



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



$\mathsf{S}(p_1) :- \mathsf{NP}(p_1, p_2, p_3), \mathsf{VP}(p_2, p_3).$

$\mathsf{VP}(p_1, p_4) :- \mathsf{V}(p_2, p_4, p_3), \mathsf{NP}(p_1, p_2, p_3).$

$\mathsf{V}(p_1, p_4, p_3) :- \mathsf{V}(p_2, p_4, p_3), \mathsf{Conj}(p_1, p_5, p_2), \mathsf{V}(p_5, p_4, p_3).$

$\mathsf{NP}(p_1, p_4, p_5) :- \mathsf{Det}(p_1, p_4, p_5, p_2, p_3), \mathsf{N}(p_2, p_3).$

$\mathsf{NP}(p_1, p_1, p_2) :- \textbf{John}(p_2).$

$\mathsf{V}(p_1, p_3, p_2) :- \textbf{find}(p_1, p_3, p_2).$

$\mathsf{V}(p_1, p_3, p_2) :- \textbf{catch}(p_1, p_3, p_2).$

$\mathsf{Conj}(p_1, p_3, p_2) :- \wedge(p_1, p_3, p_2).$

$\mathsf{Det}(p_1, p_5, p_4, p_3, p_4) :- \exists(p_1, p_2, p_4), \wedge(p_2, p_5, p_3).$

$\mathsf{N}(p_1, p_2) :- \textbf{unicorn}(p_1, p_2).$

Datalog program

$\exists(1, 2, 4).$

$\wedge(2, 5, 3).$

$\textbf{unicorn}(3, 4).$

$\textbf{find}(5, 6, 4).$

$\textbf{John}(6).$

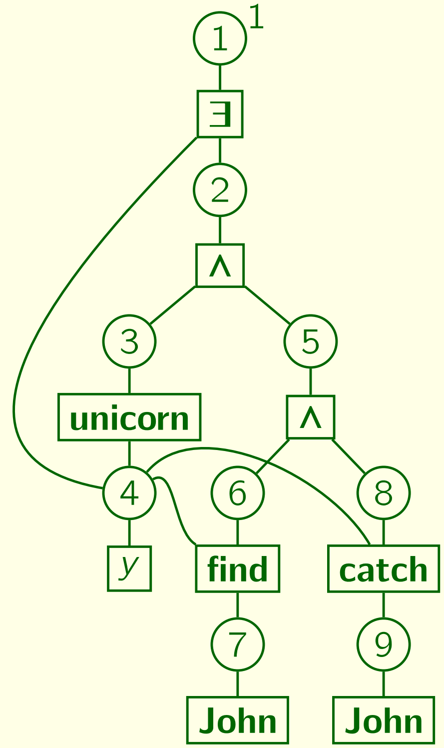
database

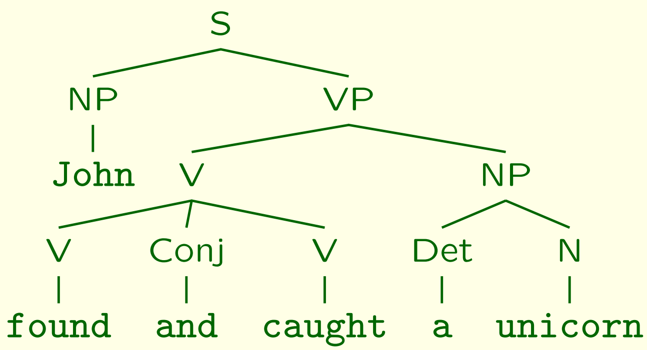
$?- \mathsf{S}(1).$

query

Reduction to Datalog

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





$\mathrm{S}(p_1)$:− $\mathrm{NP}(p_1, p_2, p_3), \mathrm{VP}(p_2, p_3)$.

$\mathrm{VP}(p_1, p_4)$:− $\mathrm{V}(p_2, p_4, p_3), \mathrm{NP}(p_1, p_2, p_3)$.

$\mathrm{V}(p_1, p_4, p_3)$:− $\mathrm{V}(p_2, p_4, p_3), \mathrm{Conj}(p_1, p_5, p_2), \mathrm{V}(p_5, p_4, p_3)$.

$\mathrm{NP}(p_1, p_4, p_5)$:− $\mathrm{Det}(p_1, p_4, p_5, p_2, p_3), \mathrm{N}(p_2, p_3)$.

$\mathrm{NP}(p_1, p_1, p_2)$:− $\textbf{John}(p_2)$.

$\mathrm{V}(p_1, p_3, p_2)$:− $\textbf{find}(p_1, p_3, p_2)$.

$\mathrm{V}(p_1, p_3, p_2)$:− $\textbf{catch}(p_1, p_3, p_2)$.

$\mathrm{Conj}(p_1, p_3, p_2)$:− $\wedge(p_1, p_3, p_2)$.

$\mathrm{Det}(p_1, p_5, p_4, p_3, p_4)$:− $\exists(p_1, p_2, p_4), \wedge(p_2, p_5, p_3)$.

$\mathrm{N}(p_1, p_2)$:− $\textbf{unicorn}(p_1, p_2)$.

Datalog program

$\exists(1, 2, 4)$.

$\wedge(2, 5, 3)$.

$\textbf{unicorn}(3, 4)$.

$\wedge(5, 8, 6)$.

$\textbf{find}(6, 7, 4)$.

$\textbf{John}(7)$.

$\textbf{catch}(8, 9, 4)$.

$\textbf{John}(9)$.

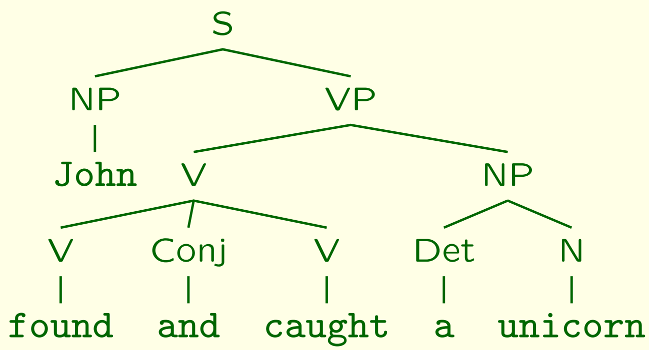
database

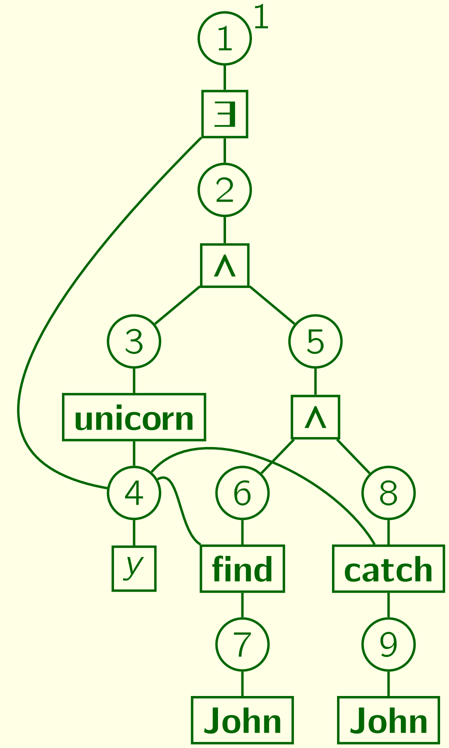
?− $\mathrm{S}(1)$.

query

Reduction to Datalog

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





$\mathrm{S}(p_1)$:− $\mathrm{NP}(p_1, p_2, p_3), \mathrm{VP}(p_2, p_3)$.

$\mathrm{VP}(p_1, p_4)$:− $\mathrm{V}(p_2, p_4, p_3), \mathrm{NP}(p_1, p_2, p_3)$.

$\mathrm{V}(p_1, p_4, p_3)$:− $\mathrm{V}(p_2, p_4, p_3), \mathrm{Conj}(p_1, p_5, p_2), \mathrm{V}(p_5, p_4, p_3)$.

$\mathrm{NP}(p_1, p_4, p_5)$:− $\mathrm{Det}(p_1, p_4, p_5, p_2, p_3), \mathrm{N}(p_2, p_3)$.

$\mathrm{NP}(p_1, p_1, p_2)$:− $\mathbf{John}(p_2)$.

$\mathrm{V}(p_1, p_3, p_2)$:− $\mathbf{find}(p_1, p_3, p_2)$.

$\mathrm{V}(p_1, p_3, p_2)$:− $\mathbf{catch}(p_1, p_3, p_2)$.

$\mathrm{Conj}(p_1, p_3, p_2)$:− $\wedge(p_1, p_3, p_2)$.

$\mathrm{Det}(p_1, p_5, p_4, p_3, p_4)$:− $\exists(p_1, p_2, p_4), \wedge(p_2, p_5, p_3)$.

$\mathrm{N}(p_1, p_2)$:− $\mathbf{unicorn}(p_1, p_2)$.

Datalog program

$\exists(1, 2, 4)$.

$\wedge(2, 5, 3)$.

$\mathbf{unicorn}(3, 4)$.

$\wedge(5, 8, 6)$.

$\mathbf{find}(6, 7, 4)$.

$\mathbf{John}(7)$.

$\mathbf{catch}(8, 9, 4)$.

$\mathbf{John}(9)$.

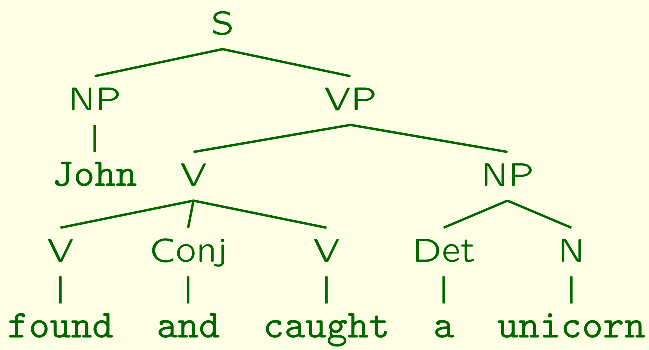
database

?− S(1).

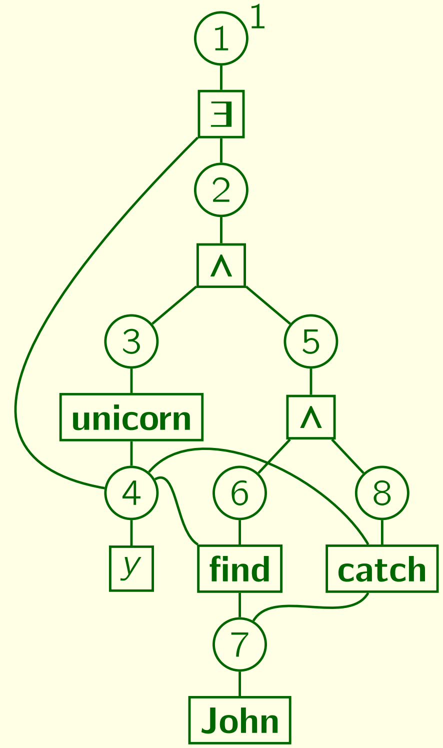
query

No!

Reduction to Datalog



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



$\mathrm{S}(p_1)$:− $\mathrm{NP}(p_1, p_2, p_3), \mathrm{VP}(p_2, p_3)$.

$\mathrm{VP}(p_1, p_4)$:− $\mathrm{V}(p_2, p_4, p_3), \mathrm{NP}(p_1, p_2, p_3)$.

$\mathrm{V}(p_1, p_4, p_3)$:− $\mathrm{V}(p_2, p_4, p_3), \mathrm{Conj}(p_1, p_5, p_2), \mathrm{V}(p_5, p_4, p_3)$.

$\mathrm{NP}(p_1, p_4, p_5)$:− $\mathrm{Det}(p_1, p_4, p_5, p_2, p_3), \mathrm{N}(p_2, p_3)$.

$\mathrm{NP}(p_1, p_1, p_2)$:− $\textbf{John}(p_2)$.

$\mathrm{V}(p_1, p_3, p_2)$:− $\textbf{find}(p_1, p_3, p_2)$.

$\mathrm{V}(p_1, p_3, p_2)$:− $\textbf{catch}(p_1, p_3, p_2)$.

$\mathrm{Conj}(p_1, p_3, p_2)$:− $\wedge(p_1, p_3, p_2)$.

$\mathrm{Det}(p_1, p_5, p_4, p_3, p_4)$:− $\exists(p_1, p_2, p_4), \wedge(p_2, p_5, p_3)$.

$\mathrm{N}(p_1, p_2)$:− $\textbf{unicorn}(p_1, p_2)$.

Datalog program

$\exists(1, 2, 4)$.

$\wedge(2, 5, 3)$.

$\textbf{unicorn}(3, 4)$.

$\wedge(5, 8, 6)$.

$\textbf{find}(6, 7, 4)$.

$\textbf{John}(7)$.

$\textbf{catch}(8, 7, 4)$.

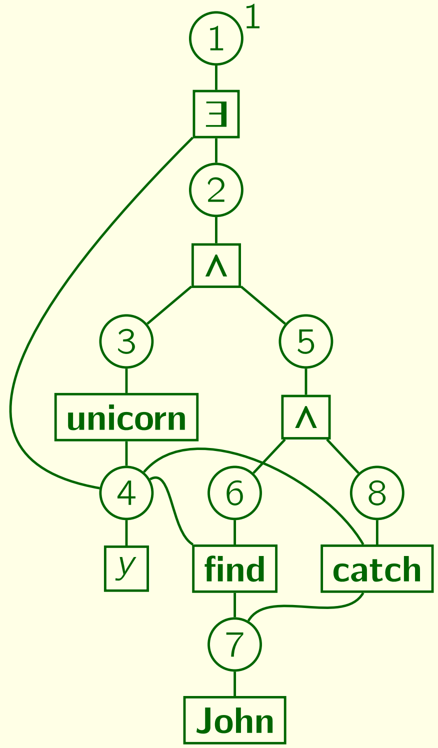
database

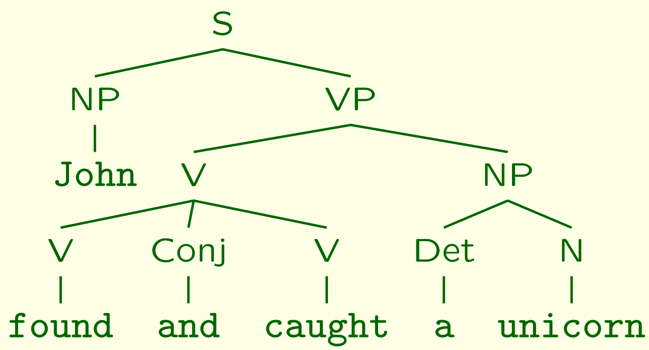
?− S(1).

query

Reduction to Datalog

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





$\mathrm{S}(p_1) :- \mathrm{NP}(p_1, p_2, p_3), \mathrm{VP}(p_2, p_3).$

$\mathrm{VP}(p_1, p_4) :- \mathrm{V}(p_2, p_4, p_3), \mathrm{NP}(p_1, p_2, p_3).$

$\mathrm{V}(p_1, p_4, p_3) :- \mathrm{V}(p_2, p_4, p_3), \mathrm{Conj}(p_1, p_5, p_2), \mathrm{V}(p_5, p_4, p_3).$

$\mathrm{NP}(p_1, p_4, p_5) :- \mathrm{Det}(p_1, p_4, p_5, p_2, p_3), \mathrm{N}(p_2, p_3).$

$\mathrm{NP}(p_1, p_1, p_2) :- \textbf{John}(p_2).$

$\mathrm{V}(p_1, p_3, p_2) :- \textbf{find}(p_1, p_3, p_2).$

$\mathrm{V}(p_1, p_3, p_2) :- \textbf{catch}(p_1, p_3, p_2).$

$\mathrm{Conj}(p_1, p_3, p_2) :- \wedge(p_1, p_3, p_2).$

$\mathrm{Det}(p_1, p_5, p_4, p_3, p_4) :- \exists(p_1, p_2, p_4), \wedge(p_2, p_5, p_3).$

$\mathrm{N}(p_1, p_2) :- \textbf{unicorn}(p_1, p_2).$

Datalog program

$\exists(1, 2, 4).$

$\wedge(2, 5, 3).$

$\textbf{unicorn}(3, 4).$

$\wedge(5, 8, 6).$

$\textbf{find}(6, 7, 4).$

$\textbf{John}(7).$

$\textbf{catch}(8, 7, 4).$

database

?− S(1).

query

Yes!

Reduction to Datalog

- Given an input λ-term, find the most compact term graph representing it.

From Datalog derivation to grammar derivation

- $\mathsf{S}(1)$
  - $\mathsf{NP}(1,1,6)$
    - $\textbf{John}(6)$
  - $\mathsf{VP}(1,6)$
    - $\mathsf{V}(5,6,4)$
      - $\textbf{find}(5,6,4)$
    - $\mathsf{NP}(1,5,4)$
      - $\mathsf{Det}(1,5,4,3,4)$
        - $\exists(1,2,4)$
        - $\wedge(2,5,3)$
      - $\mathsf{N}(3,4)$
        - $\textbf{unicorn}(3,4)$

$$\Downarrow$$

- $\mathsf{S}((\lambda u.u\ \textbf{John})(\lambda x.(\lambda uv.\exists(\lambda y.\wedge(uy)(vy))\ \textbf{unicorn}\ (\lambda y.\textbf{find}\ y\ x))))$
  - $\mathsf{NP}(\lambda u.u\ \textbf{John})$
  - $\mathsf{VP}(\lambda x.(\lambda uv.\exists(\lambda y.\wedge(uy)(\lambda y.vy))\ \textbf{unicorn}\ (\lambda y.\textbf{find}\ y\ x)))$
    - $\mathsf{V}(\textbf{find})$
    - $\mathsf{NP}(\lambda uv.\exists(\lambda y.\wedge(uy)(vy))\ \textbf{unicorn})$
      - $\mathsf{Det}(\lambda uv.\exists(\lambda y.\wedge(uy)(\lambda y.vy)))$
      - $\mathsf{N}(\textbf{unicorn})$

$$\textbf{John} : 6, \textbf{find} : 4 \to 6 \to 5, \exists : (4 \to 2) \to 1, \wedge : 3 \to 5 \to 2, \textbf{unicorn} : 4 \to 3$$
$$\vdash (\lambda u.u\ \textbf{John})(\lambda x.(\lambda uv.\exists(\lambda y.\wedge(uy)(vy))\ \textbf{unicorn}\ (\lambda y.\textbf{find}\ y\ x))) : 1$$

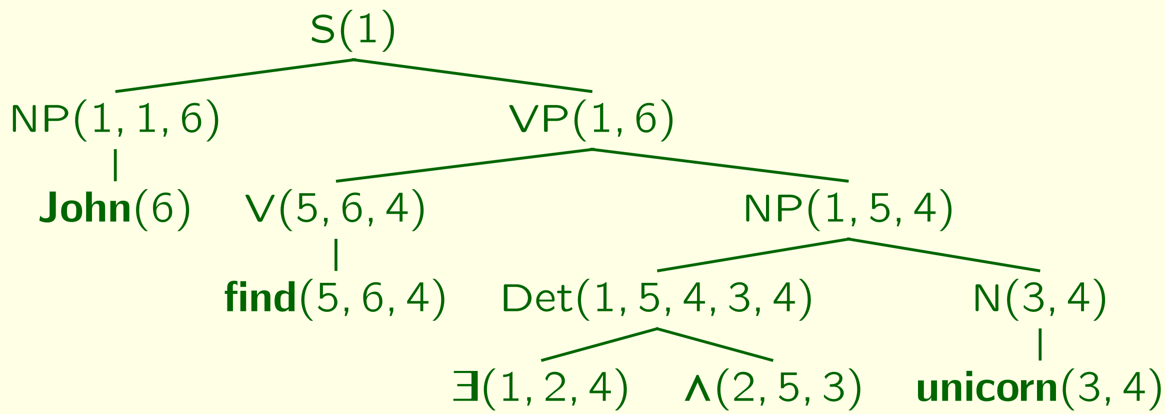
Aoto and Ono's Coherence Theorem $\Downarrow$

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

$$\Downarrow$$

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

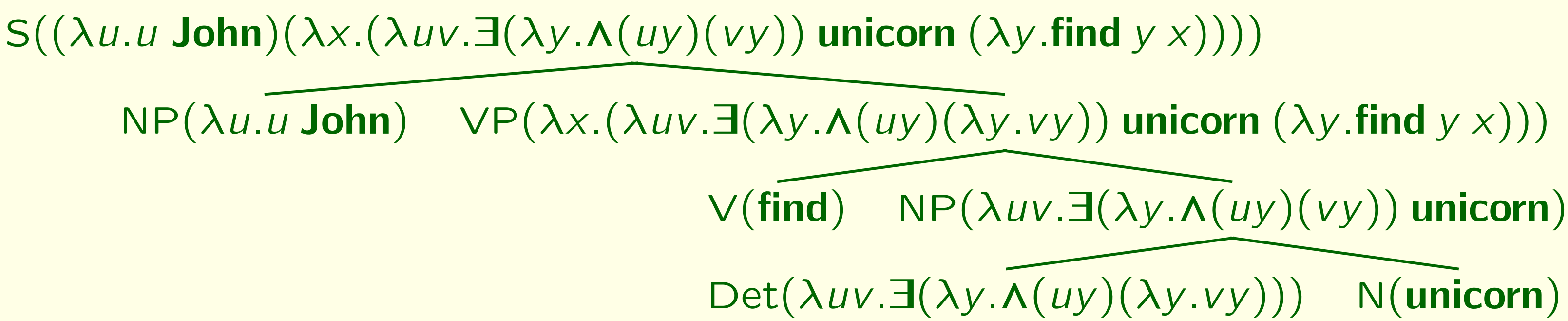
From grammar derivation to Datalog derivation



$$\Uparrow$$

$$\mathbf{John} : 6, \mathbf{find} : 4 \to 6 \to 5, \exists : (4 \to 2) \to 1, \wedge : 3 \to 5 \to 2, \mathbf{unicorn} : 4 \to 3$$
$$\vdash (\lambda u.u\ \mathbf{John})(\lambda x.(\lambda uv.\exists(\lambda y.\wedge(uy)(vy))\ \mathbf{unicorn}\ (\lambda y.\mathbf{find}\ y\ x))) : 1$$

Subject Expansion $\Uparrow$



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

$$\Uparrow$$

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

Tight complexity bound

- "ϵ-rule":

$$B(M)$$

  where M is a combinator.

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

Tight complexity bound

- "ϵ-rule":

$$B(M)$$

where M is a combinator.

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

- If G is an almost linear CFLG without ϵ-rules and M is the normal form of an almost linear λ-term, any Datalog derivation tree for

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

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

Tight complexity bound

- "ϵ-rule":

  $$B(M)$$

  where M is a combinator.

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

- If G is an almost linear CFLG without ϵ-rules and M is the normal form of an almost linear λ-term, any Datalog derivation tree for

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

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

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

Tight complexity bound

- "ϵ-rule":

$$B(M)$$

  where M is a combinator.

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

- If G is an almost linear CFLG without ϵ-rules and M is the normal form of an almost linear λ-term, any Datalog derivation tree for

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

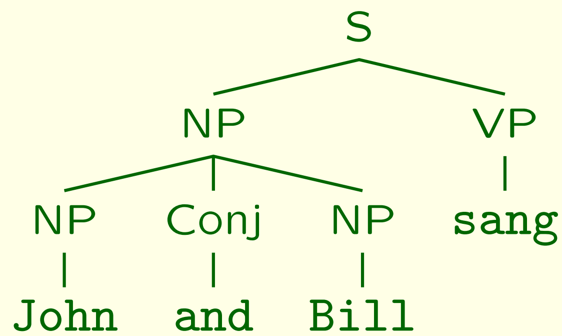
  has a number of leaves linear in the size of M.

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

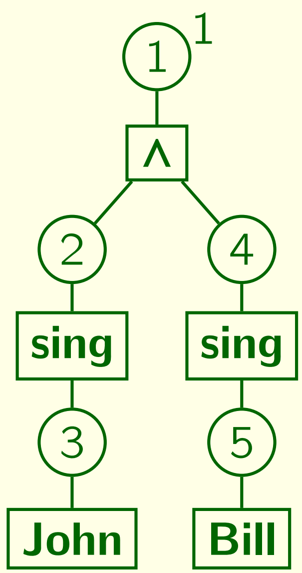
- This implies that $L(G)$ belongs to LOGCFL (cf. Ullman and Van Gelder 1988, Kanellakis 1988).

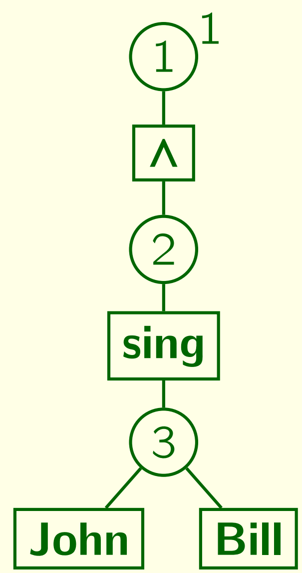
Limitation

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



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





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 λ-terms need not be avoided for the purpose of achieving computational efficiency.