Advances in Abstract Categorial Grammars
Language Theory and Linguistic Modeling

Lecture 3

Reduction of second-order ACGs to Datalog
Extension to “almost linear” second-order ACGs
CFG recognition/parsing

S → NP VP
VP → V NP
NP → Det N
NP → John
V → found
Det → a
N → unicorn

John found a unicorn ∈ L(G) ?

To give an idea of what the reduction looks like
Well-known case of CFGs
Datalog query evaluation

\begin{align*}
S(i, k) & : \text{ NP}(i, j), \text{ VP}(j, k). \\
\text{VP}(i, k) & : \text{ V}(i, j), \text{ NP}(j, k). \\
\text{NP}(i, k) & : \text{ Det}(i, j), \text{ N}(j, k). \\
\text{NP}(i, j) & : \text{ John}(i, j). \\
\text{V}(i, j) & : \text{ found}(i, j). \\
\text{Det}(i, j) & : \text{ a}(i, j). \\
\text{N}(i, j) & : \text{ unicorn}(i, j).
\end{align*}

\text{John}(0, 1). \\
\text{found}(1, 2). \\
\text{a}(2, 3). \\
\text{unicorn}(3, 4). \\
?- S(0, 4). \\

\text{Definite clause grammar representation} \\
\text{Executable as Prolog code}
The conversion is very straightforward
String -> string graph

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

S(i, k) :- NP(i, j), VP(j, k).

?- S(0, 4).
CFG recognition/parsing ≈ Datalog query evaluation

CFG derivation tree and Datalog derivation tree isomorphic to each other
Finding one amounts to finding the other
The Datalog representation extends to various grammars through (almost linear) second-order ACGs
Need non-linear terms to represent logical formulas
Parsing and generation as Datalog query evaluation

- Algorithms
  - Seminaive bottom-up ≈ CYK
  - Magic-sets rewriting ≈ Earley

- Computational complexity
  - Fixed grammar recognition
  - Uniform recognition
  - Parsing

Allows a uniform approach to parsing and generation
Sophisticated evaluation methods for Datalog apply to parsing/generation
Polynomial-time algorithm

\[
\text{Seminaive}(\mathbf{P}, \mathbf{D})
\]

1. \( \text{agenda}[0] \leftarrow \mathbf{D} \)
2. \( i \leftarrow 0 \)
3. \( \text{chart} \leftarrow \emptyset \)
4. \( \text{while} \ \text{agenda}[i] \neq \emptyset \)
5. \( \text{do} \)
6. \( \text{chart} \leftarrow \text{chart} \cup \text{agenda}[i] \)
7. \( \text{agenda}[i + 1] \leftarrow \text{Conseq}(\mathbf{P}, \text{agenda}[i], \text{chart}) \)
8. \( i \leftarrow i + 1 \)
9. \( \text{return} \ \text{chart} \)

- holds facts with derivation tree of minimal height \( i \)
- facts that immediately follow from one fact in \( \text{agenda}[i] \) plus some facts in \( \text{chart} \)

\( \approx \) well-formed substring table

Works for Datalog programs in general
Parsing algorithms are often not explicitly stated in textbooks.
Computational complexity

- CFG

<table>
<thead>
<tr>
<th>Fixed grammar recognition</th>
<th>LOGCFL-complete</th>
</tr>
</thead>
<tbody>
<tr>
<td>Uniform recognition</td>
<td>P-complete</td>
</tr>
<tr>
<td>ε-free uniform recognition</td>
<td>LOGCFL-complete</td>
</tr>
</tbody>
</table>

Same holds for classes of grammars that can be represented by Datalog programs of bounded degree
Computational complexity

- Almost linear second-order ACGs with **bounded width and rank**

<table>
<thead>
<tr>
<th>Fixed grammar recognition</th>
<th>LOGCFL-complete</th>
</tr>
</thead>
<tbody>
<tr>
<td>Uniform recognition</td>
<td>P-complete</td>
</tr>
<tr>
<td>ε-free uniform recognition</td>
<td>LOGCFL-complete</td>
</tr>
</tbody>
</table>

width = |σ(B)| = arity of B in Datalog
rank = number of subgoals
ε-rule = Datalog rule with empty right-hand side

ε-rule = rule whose right-hand side is empty and whose left-hand side argument is a pure λ-term
LOGCFL

- The class of problems that reduce to some context-free language in logarithmic space
- The smallest computational complexity class that includes the context-free languages

\[ AC^0 \subseteq NC^1 \subseteq L \subseteq NL \subseteq LOGCFL \subseteq AC^1 \subseteq NC^2 \subseteq P \subseteq NP \]

A very important complexity class
Implies the existence of efficient parallel algorithm
Datalog in computational linguistics

- Definite Clause Grammar
  - Pereira and Warren 1980

- Deduction system
  - Shieber et al. 1997

- Uninstantiated parsing system
  - Sikkel 1997

\[
\begin{align*}
\text{[NP, } i, j \text{]} & \quad \text{[VP, } j, k \text{]} \\
\text{[S, } i, k \text{]} \\
\end{align*}
\]

The idea of using Datalog is not new. DCG is too powerful. Datalog notation is more convenient.
CFG + Montague semantics

One way of writing Montague semantics with CFG:
Grammar rules associates a lambda-term to each node
Must reduce to normal form to get the desired representation
John found a unicorn

\[ \exists y (\text{unicorn}(y) \land \text{find}(\text{John}, y)) \]

**Input logical form**

**Derivation tree**

**Output surface form**

Tactical generation or surface realization
Parsing of input logical form

$\exists y (\text{unicorn}(y) \land \text{find}(\text{John}, y))$

Can concentrate on the semantic half of the grammar
Recognition of surface realizability

\[ \exists y (\text{unicorn}(y) \land \text{find}(\text{John}, y)) \]

input logical form

surface realizable?

yes/no

Solving this problem almost amounts to solving surface realization
Context-free grammar on \( \lambda \)-terms

\[
S(X_1 X_2) : \rightarrow NP(X_1), VP(X_2)
\]
\[VP(\lambda x. X_2 (\lambda y. X_1 yx)) : \rightarrow V(X_1), NP(X_2).
\]
\[V(\lambda yx. X_2 (X_1 yx) (X_3 yx)) : \rightarrow V(X_1), Conj(X_2), V(X_3).
\]
\[NP(X_1 X_2) : \rightarrow Det(X_1), N(X_2).
\]
\[NP(\lambda u. u John^e).
\]
\[V(find^{e \rightarrow e \rightarrow t}).
\]
\[V(catch^{e \rightarrow e \rightarrow t}).
\]
\[Conj(^{t \rightarrow t \rightarrow t}).
\]
\[Det(\lambda u v. \exists^{(e \rightarrow t) \rightarrow t} (\lambda y. ^{t \rightarrow t \rightarrow t} (uy)(vy))).
\]
\[N(unicorn^{e \rightarrow t}).
\]

CFG + Montague – CFG = CFLG
Generates a set of lambda-terms
Context-free grammar on \( \lambda \)-terms

\[
\begin{align*}
S(X_1X_2) &\coloneqq NP(X_1), VP(X_2) \\
VP(\lambda x.X_2(\lambda y.X_1yx)) &\coloneqq V(X_1), NP(X_2) \\
V(\lambda yx.X_2(X_1yx)(X_3yx)) &\coloneqq V(X_1), Conj(X_2), V(X_3) \\
NP(X_1X_2) &\coloneqq Det(X_1), N(X_2) \\
NP(\lambda uu \text{ John}^e) &. \\
V(\text{find}^{e\to e\to t}) &. \\
V(\text{catch}^{e\to e\to t}) &. \\
Conj(\wedge^{t\to t\to t}) &. \\
Det(\lambda uv.\exists^{e\to t}(\lambda y.\wedge^{t\to t\to t}(uy)(vy))) &. \\
N(u\text{nicorn}^{e\to t}) &. \\
\end{align*}
\]

\[
\begin{align*}
\sigma(S) &= t \\
\sigma(VP) &= e \to t \\
\sigma(NP) &= (e \to t) \to t \\
\sigma(V) &= e \to e \to t \\
\sigma(Conj) &= t \to t \to t \\
\sigma(Det) &= (e \to t) \to (e \to t) \to t \\
\sigma(N) &= e \to t
\end{align*}
\]

A nonterminal is associated with a type. Arguments are \( \lambda \)-terms of that type.
Context-free grammars on $\lambda$-terms = second-order non-linear ACGs

$\pi : B(M) \Rightarrow B_1(X_1), \ldots, B_n(X_n)$. 

$\pi : B_1 \rightarrow \cdots \rightarrow B_n \rightarrow B$

$\lambda X_1 \ldots X_n . M$

- type of $\pi$
- object realization of $\pi$
- abstract constant
A pair of second-order (non-linear) ACGs as a “synchronous” grammar.
From second-order ACG recognition to Datalog query evaluation

S(X₁,X₂) := NP(X₁) VP(X₂)  
VP(λx.X₂(λy.X₁yx)) := V(X₁), NP(X₂).  
V(λyx.X₂(X₁yx)(X₃yx)) := V(X₁), Conj(X₂), V(X₃).  
NP(X₁,X₂) := Det(X₁), N(X₂).  
NP(λu.u John).  
V(find).  
V(catch).  
Conj(∧).  
Det(λuv.∃(e→e→t)(λy.∧(t→t→t)(uy)(vy))).  
N(unicorn).  

∃(λy.∧(unicorn)(find y John)) ∈ L(G)?
From second-order ACG recognition to Datalog query evaluation

\[ S(i) \leftarrow \text{NP}(i_1, i_2, i_3) \quad \text{VP}(i_2, i_3) \]
\[ \text{VP}(i, i_4) \leftarrow \text{V}(i_2, i_4, i_3), \text{NP}(i_1, i_2, i_3). \]
\[ \text{V}(i_1, i_4, i_3) \leftarrow \text{V}(i_2, i_4, i_3), \text{Conj}(i_1, i_5, i_2), \text{V}(i_5, i_4, i_3). \]
\[ \text{NP}(i_1, i_4, i_5) \leftarrow \text{Det}(i_1, i_4, i_5, i_2, i_3), \text{N}(i_2, i_3). \]
\[ \text{NP}(i_1, i_1, i_2) \leftarrow \text{John}(i_2). \]
\[ \text{V}(i_1, i_3, i_2) \leftarrow \text{find}(i_1, i_3, i_2). \]
\[ \text{V}(i_1, i_3, i_2) \leftarrow \text{catch}(i_1, i_3, i_2). \]
\[ \text{Conj}(i_1, i_3, i_2) \leftarrow \land(i_1, i_3, i_2). \]
\[ \text{Det}(i_1, i_5, i_4, i_3, i_4) \leftarrow \exists(i_1, i_2, i_4), \land(i_2, i_5, i_3). \]
\[ \text{N}(i_1, i_2) \leftarrow \text{unicorn}(i_1, i_2). \]

Given conversion to Datalog, can use general Datalog techniques.
John(0, 1)  found(1, 2)  a(2, 3)  unicorn(3, 4)

\[ S(i, k) :- \text{NP}(i, j), \text{VP}(j, k). \]

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

The way the program, the database, and the query are obtained is similar to the CFG case. Objects derived by the grammar are represented by (hyper)graphs.
\[ \exists(\lambda y . \land (uy)(vy)) \]

\[ \lambda u v . \exists(\lambda y . \land (uy)(vy)) \]

\[ \exists \]
\[ \land \]
\[ \lambda y \]
\[ i_1 \]
\[ i_2 \]
\[ i_3 \]
\[ i_4 \]
\[ i_5 \]
\[ i_6 \]

\[ u \]
\[ v \]
\[ y \]

\[ \text{tree graph} \]

\[ \text{term graph} \]

\[ \text{term graph with external nodes} \]

\[ \lambda \text{-terms can also be represented by hypergraphs (when almost linear).} \]

\[ \text{A hypergraph is a “term graph” when each node is the “result node” of a unique hyperedge.} \]

\[ \text{A hyperedge is “directed”: the nodes it attaches to are ordered.} \]
\[ \text{Det}(\lambda uv. \exists^{(e \rightarrow t) \rightarrow t} (\lambda y. \land^{t \rightarrow t \rightarrow t} (uy)(vy))). \]

How a rule is converted to a Datalog rule.
External nodes become arguments of the head.
Edges labeled by constants become subgoals.
Edges labeled by free variables (none in this example) become subgoals.
How the input λ-term is converted to database and query.
Edges labeled by constants constitute the database.
External nodes become arguments of the query.
From ACG recognition to Datalog query evaluation

- The reduction is correct when all λ-terms in the grammar are **almost linear**.

\[
\begin{align*}
\Gamma \vdash x : \alpha & \quad \Gamma \vdash c : \tau(c) \\
\Gamma \vdash M : \alpha & \quad \Delta \vdash N : \beta \\
\Gamma \cup \Delta \vdash MN : \beta & \quad \text{if } \Gamma \cap \Delta \subseteq \text{At} \\
\Gamma, x : \alpha \vdash M : \beta & \\
\Gamma \vdash \lambda x. M : \alpha \to \beta
\end{align*}
\]

almost affine

Vacuous abstraction is not allowed.
If a variable occurs twice in a subterm, it must have an atomic type.
Context-free grammar on $\lambda$-terms

$$S(X_1X_2) := \text{NP}(X_1), \text{VP}(X_2)$$

$$\text{VP}(\lambda x.X_2(\lambda y.X_1yx)) := \text{V}(X_1), \text{NP}(X_2).$$

$$\text{V}(\lambda y^e x^e.X_2(X_1y^e x^e)(X_3y^e x^e)) := \text{V}(X_1), \text{Conj}(X_2), \text{V}(X_3).$$

$$\text{NP}(X_1X_2) := \text{Det}(X_1), \text{N}(X_2).$$

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

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

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

$$\text{Conj}(\land^{t\rightarrow t\rightarrow t}).$$

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

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

All $\lambda$-terms almost linear.
Generates $\beta$–normal forms of almost linear $\lambda$–terms.
Almost linear $\lambda$-terms

- The class of almost linear $\lambda$-terms is not closed under $\beta$-reduction.

$$(\lambda x^e . y^{e \rightarrow e \rightarrow t} x x)(z^{e \rightarrow e} w^e) \rightarrow_\beta y(zw)(zw)$$
A formal definition of the hypergraph associated with an almost linear λ-term. The input may have to be β-expanded first. The graph of a constant or variable of type α has |α| nodes, all of which are external.
For $M^{\alpha \rightarrow \beta}N^\alpha$, the last $|\alpha|$ external nodes of graph(M) are identified with the external nodes of graph(N).
The new external nodes are the remaining external nodes of graph(M).
The edges labeled by the same variable (and the nodes they attach to) are also merged. Such a variable is atomic-typed.
The nodes that the abstracted variable attaches to are appended to the list of external nodes.
\[ M = \lambda uv. \exists^{(e \to t) \to t} (\lambda y. \land^{t \to t \to t} (uy)(vy)) \]

\[
\begin{align*}
\exists &: (e \to t) \to t, \land : t \to t \to t \vdash M : (e \to t) \to (e \to t) \to t \\
\exists &: (i_4 \to i_2) \to i_1, \land : i_3 \to i_5 \to i_2 \vdash M : (i_4 \to i_3) \to (i_4 \to i_5) \to i_1
\end{align*}
\]

**principal typing**

The construction of the graph gives a principal typing (i.e., most general typing).
Typed \( \lambda \)-calculus

- If \( M \) is typable, \( M \) has a unique principal typing.
- Subject Reduction:

\[
M \xrightarrow{\beta} M' \quad \text{and} \quad \Gamma \vdash M : \alpha \quad \iff \quad \Gamma \vdash M' : \alpha
\]

General facts of importance.
All other typings are instantiations of the principal typing.
Pure linear $\lambda$-terms

- The principal typing of an affine $\lambda$-term is balanced. \textit{Belnap 1976}

- If $\mathcal{M}$ has a balanced typing, it is affine. \textit{Hirokawa 1992}

$$X_1 : \vdash i_3 \to i_4 \to \overline{i_2}, \quad X_2 : (i_3 \to i_2) \to i_1 \vdash \lambda x. X_2 (\lambda y. X_1 y x) : i_4 \to i_1$$

Many properties of linear $\lambda$-terms carry over to almost linear. Linear = affine + $\lambda I$

“Balanced” means that there is at most one positive and at most one negative occurrence of any atomic type.
Pure linear $\lambda$-terms

- **Coherence Theorem.** All inhabitants of a balanced typing are $\beta\eta$-equal.

  Babaev and Solov’ev 1979
Pure linear $\lambda$-terms

- Subject Expansion Theorem.

$M \xrightarrow{\beta} M'$ and $\Gamma \vdash M' : \alpha \implies \Gamma \vdash M : \alpha$

non-erasing
non-duplicating

Hindley

$(\lambda y.x(y(\lambda z.z))y)(\lambda w.w)$

$x : (i \rightarrow i) \rightarrow (j \rightarrow j) \rightarrow k \vdash x(\lambda z.z)(\lambda w.w) : k$
Pure almost linear \( \lambda \)-terms

- The principal typing of an \textbf{almost affine} \( \lambda \)-term is \textbf{negatively non-duplicated}.

- If \( M \) has a negatively non-duplicated typing, it is \( \beta \eta \)-equal to an almost affine \( \lambda \)-term.

\[
M = \lambda uv. \exists (e \to t) \to t (\lambda y. \land t \to t (uy)(vy))
\]

Almost linear = almost affine + \( \lambda I \)

“Negatively non–duplicated” means that there is at most one negative occurrence of any atomic type.
Pure almost linear $\lambda$-terms

- **Coherence Theorem.** All inhabitants of a negatively non-duplicated typing are $\beta\eta$-equal.

  Aoto and Ono 1994
Pure almost linear $\lambda$-terms

- **Subject Expansion Theorem.**
  \[ M \xrightarrow{\beta} M' \text{ and } \Gamma \vdash M' : \alpha \iff \Gamma \vdash M : \alpha \]
  non-erasing
  almost non-duplicating

- The leftmost reduction from an almost affine $\lambda$-term is almost non-duplicating.

  \[
  (\lambda w. (\lambda x. yxx)(wz))(\lambda v. v) \\
  (\lambda x. yxx)((\lambda v. v)z) \quad (\lambda w. y(wz)(wz))(\lambda v. v)
  \]

A reduction is “almost non-duplicating” if any duplicating contraction involves $\lambda$ binding an atomic typed variable.
\[ VP(\lambda x. X_2(\lambda y. X_1 y x)) : - V(X_1), NP(X_2). \]

\[ VP(\lambda y x. X_2(X_1 y x)(X_3 y x)) : - V(X_1), Conj(X_2), V(X_3). \]

\[ \exists(\lambda y. \land(\text{unicorn } y)(\text{find } y \text{ John})) \]

**Datalog program**

\[
\begin{align*}
S(i_1) & : - NP(i_1, i_2, i_3), VP(i_2, i_3). \\
VP(i_1, i_4) & : - V(i_2, i_4, i_3), NP(i_1, i_2, i_3). \\
V(i_1, i_4, i_3) & : - V(i_2, i_4, i_3), Conj(i_1, i_5, i_2), V(i_5, i_4, i_3). \\
NP(i_1, i_4, i_5) & : - Det(i_1, i_4, i_5, i_2, i_3), N(i_2, i_3). \\
NP(i_1, i_1, i_2) & : - \text{John}(i_2). \\
V(i_1, i_3, i_2) & : - \text{find}(i_1, i_3, i_2). \\
V(i_1, i_3, i_2) & : - \text{catch}(i_1, i_3, i_2). \\
Conj(i_1, i_3, i_2) & : - \land(i_1, i_3, i_2). \\
Det(i_1, i_5, i_4, i_3, i_4) & : - \exists(i_1, i_2, i_4), \land(i_2, i_5, i_3). \\
N(i_1, i_2) & : - \text{unicorn}(i_1, i_2).
\end{align*}
\]
A tricky case.
\[ \exists (\lambda y. \left( \text{unicorn } y \right) \land (\text{find } y \text{ John}) \land (\text{catch } y \text{ John})) \]

**Datalog program**

1. \[ S(t_1) : \neg \text{NP}(t_1, t_2, t_3), \text{VP}(t_2, t_3). \]
2. \[ \text{VP}(t_1, i_4) : \neg \text{V}(t_2, i_4, t_3), \text{NP}(t_1, i_2, i_3). \]
3. \[ \text{V}(t_1, i_4, t_3) : \neg \text{V}(t_2, i_4, t_3), \text{Conj}(t_1, i_5, i_2), \text{V}(t_5, i_4, i_3). \]
4. \[ \text{NP}(t_1, i_4, i_5) : \neg \text{Det}(t_1, i_4, i_5, i_2, i_3), \text{N}(t_5, i_3). \]
5. \[ \text{NP}(t_1, i_1, i_2) : \neg \text{John}(i_2). \]
6. \[ \text{V}(t_1, i_3, i_2) : \neg \text{find}(t_1, i_3, i_2). \]
7. \[ \text{V}(t_1, i_3, i_2) : \neg \text{catch}(t_1, i_3, i_2). \]
8. \[ \text{Conj}(t_1, i_3, i_2) : \neg \land(t_1, i_3, i_2). \]
9. \[ \text{Det}(t_1, i_5, i_4, i_3, i_4) : \neg \exists(t_1, i_2, i_4), \land(t_2, i_5, i_3). \]
10. \[ \text{N}(t_1, i_2) : \neg \text{unicorn}(t_1, i_2). \]

**Database**

- \[ \exists(1, 2, 4). \]
- \[ \land(2, 5, 3). \]
- \[ \text{unicorn}(3, 4). \]
- \[ \land(5, 8, 6). \]
- \[ \text{find}(6, 7, 4). \]
- \[ \text{John}(7). \]
- \[ \text{catch}(8, 7, 4). \]

**Query**

\[ ?- S(1). \]

**Yes!**
Given an input \( \lambda \)-term, find the most compact term graph (\( \approx \) fully collapsed form) representing it.

\[
M = \exists(\lambda y. \exists(\textsf{unicorn } y)(\exists(\textsf{find y John})(\textsf{catch y John})))
\]

This graph represents a pure almost linear \( \lambda \)-term.

\[
M' = (\lambda x. \exists(\lambda y. \exists(\textsf{unicorn } y)(\exists(\textsf{find } y x)(\exists(\textsf{catch } y x))))) \textsf{ John}
\]

The two occurrences of John are identified, but not the two occurrences of \( \exists \).
Reduction to Datalog

\[
\begin{align*}
\exists (1, 2, 4). \\
\land (2, 5, 3). \\
\text{unicorn}(3, 4). \\
\land (5, 8, 6). \\
\text{find}(6, 7, 4). \\
\text{John}(7). \\
\text{catch}(8, 7, 4). \\
? \leftarrow S(1).
\end{align*}
\]

• (database($M'$), query($M'$)) represents a set of \(\lambda\)-terms.
\[
\mathcal{L} = \{ N' | \exists : (4 \rightarrow 2) \rightarrow 1, \land_1 : 3 \rightarrow 5 \rightarrow 2, \text{unicorn} : 4 \rightarrow 3, \\
\land_2 : 6 \rightarrow 8 \rightarrow 5, \text{find} : 6 \rightarrow 7 \rightarrow 5, \text{John} : 7 \rightarrow N' : 1 \}
\]

• All elements of \(\mathcal{L}\) are \(\beta\eta\)-equal by Aoto and Ono’s Coherence Theorem.

• This set contains all almost linear \(\lambda\)-terms that \(\beta\)-reduce to \(|M'|_\beta\) by the Subject Expansion Theorem.

• But this is not enough!
Reduction to Datalog

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

- (database(M'), query(M')) represents a set of \(\lambda\)-terms. \[ \mathcal{L} = \{N' | \exists : (4 \rightarrow 2) \rightarrow 1, \land_1 : 3 \rightarrow 5 \rightarrow 2, \text{unicorn} : 4 \rightarrow 3, \land_2 : 6 \rightarrow 8 \rightarrow 5, \text{find} : 6 \rightarrow 7 \rightarrow 5, \text{John} : 7 \vdash N' : 1 \} \]
- Since M' is the most compact almost linear \(\lambda\)-term such that \(M'\theta \rightarrow^\beta M\) (where \(\theta\) is the substitution that gives back the original constants), for every almost linear \(N\) such that \(N \rightarrow^\beta M\), there is an \(N' \in \mathcal{L}\) such that \(N'\theta = N\).
\(\exists (\lambda y. \wedge(\text{unicorn } y)(\wedge(\text{find } y \text{ John})(\text{catch } y \text{ John})))\)

\[ N[\wedge, \wedge, \text{John}] \rightarrow M[\wedge, \wedge, \text{John}, \text{John}] \]

\[ N'[\wedge_1, \wedge_2, \text{John}] \rightarrow M[\wedge_1, \wedge_2, \text{John}, \text{John}] \]

\[ M'[\wedge_1, \wedge_2, \text{John}] \]

\((\lambda x. \exists (\lambda y. \wedge_1(\text{unicorn } y)(\wedge_2(\text{find } y \ x)(\text{catch } y \ x))))\) \text{ John} \)
A Datalog derivation tree determines a grammar derivation plus a typing of the associated \( \lambda \)-term.
Limitations

NP(λx→t. X_2(X_1x)(X_3x)) :— NP(X_1), Conj(X_2), NP(X_3).

\[ \land(\text{sing John})(\text{sing Bill}) \]

\[ 1 \quad \land \quad 2 \]

\[ \quad \text{sing} \quad \text{sing} \]

\[ 3 \quad 5 \]

\[ \text{John} \quad \text{Bill} \]

not a term graph
Regular sets as input

- For a linear grammar $G$, (database($A$), query($A$)) representing a finite (string or tree) automaton can be used with program($G$).
Regular sets as input

- For a tree generating almost linear grammar $G$, (database($A$), query($A$)) representing a deterministic bottom-up finite tree automaton $A$ can be used with program($G$).
  - PMCFG recognition via PMRTG
  - generation from regular sets as underspecified representations