Advances in Abstract Categorial Grammars

Language Theory and Linguistic Modeling

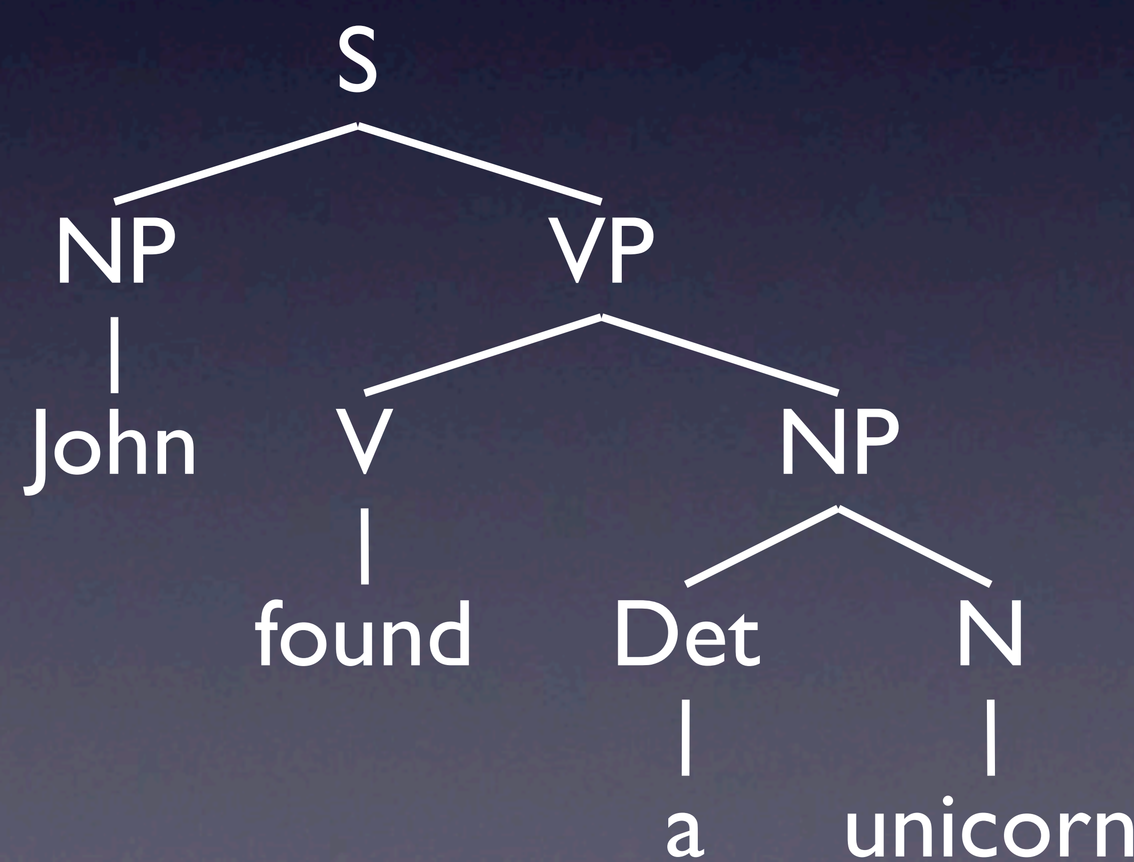
Lecture 3

Reduction of second-order ACGs to 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

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

program

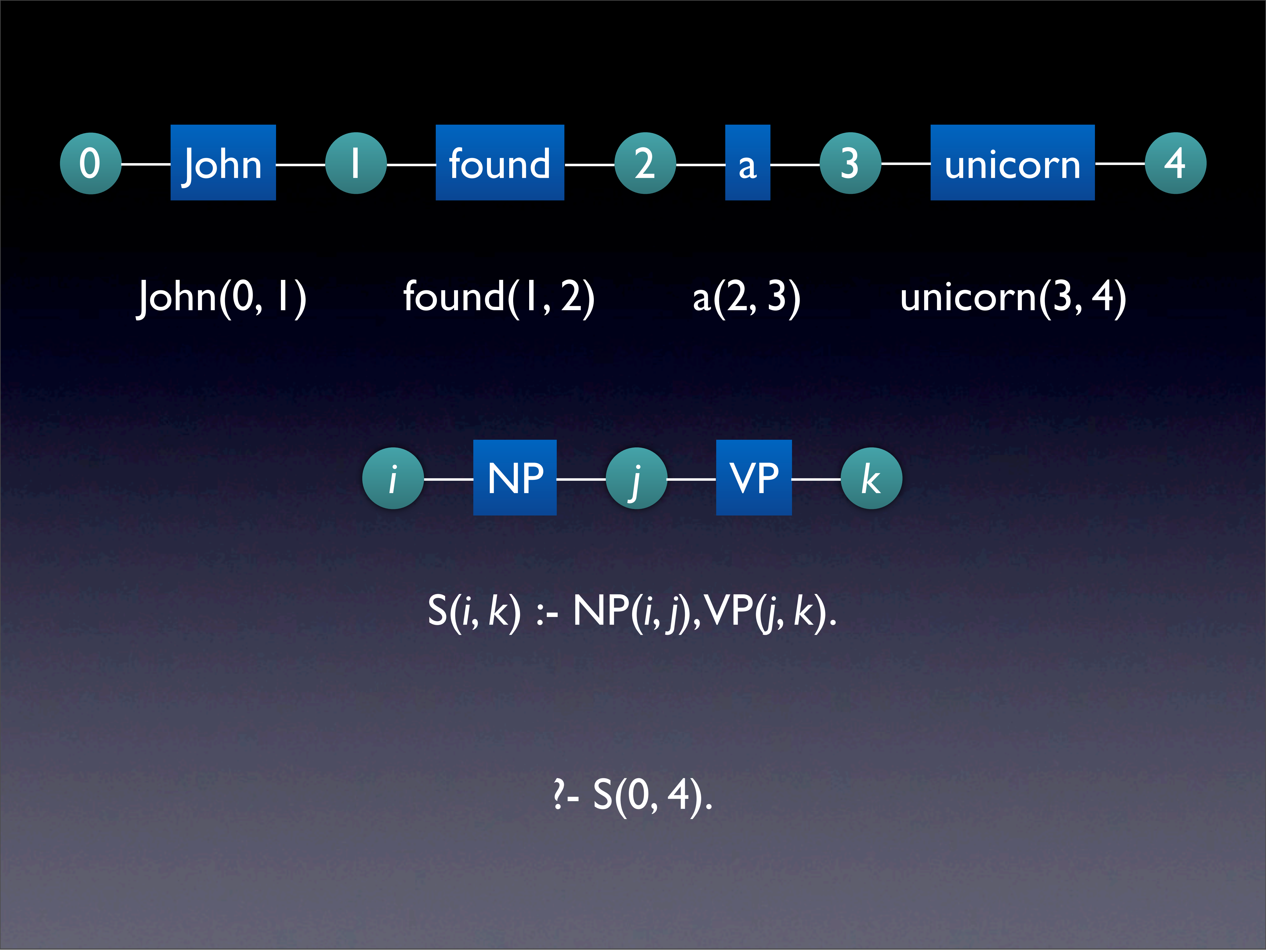
```
John(0, 1).
found(1, 2).
a(2, 3).
unicorn(3, 4).
```

database

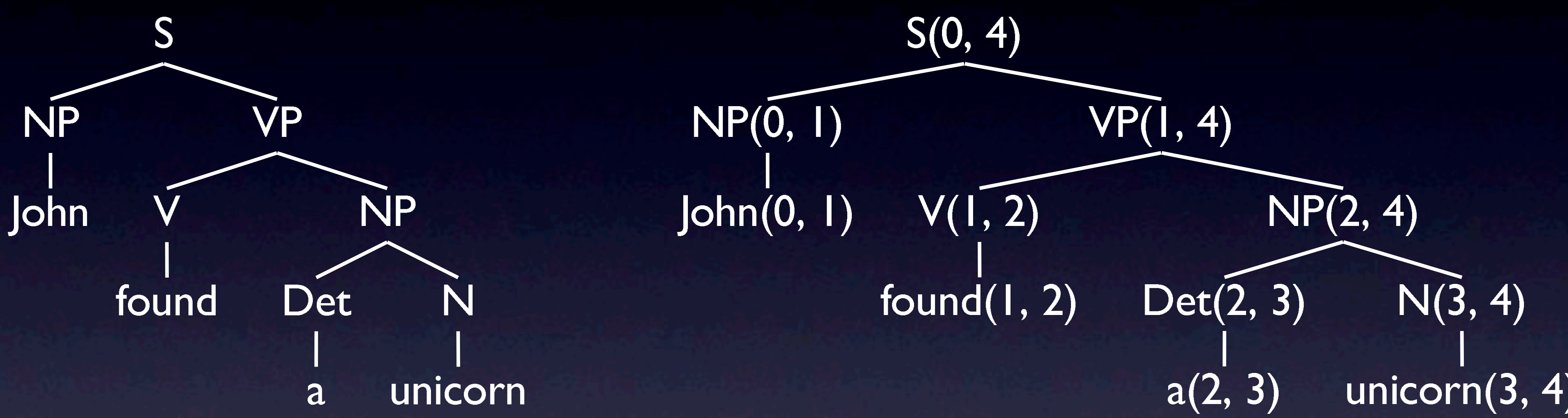
```
?- S(0, 4).
```

query

Definite clause grammar representation
Executable as Prolog code



The conversion is very straightforward
String -> string graph



CFG recognition/parsing ≈ Datalog query evaluation

CFG derivation tree and Datalog derivation tree isomorphic to each other
Finding one amounts to finding the other

Parsing and generation as Datalog query evaluation

Kanazawa 2007

Recognition/Parsing	CFG MCFG RTG MRTG $\mathrm{CFTG}_{\mathrm{IO}}$ ⋮
Generation	CFG, etc. + Montague semantics†

→ **Recognition/Parsing**: almost linear second-order ACGs → Datalog query evaluation

† when almost linear

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

Seminaive$(\mathbf{P}, D)$

$$
\begin{array}{l}
\mathit{agenda}[0] \leftarrow D \\
i \leftarrow 0 \\
\mathit{chart} \leftarrow \varnothing \\
\textbf{while } \mathit{agenda}[i] \neq \varnothing \\
\quad\quad \textbf{do} \\
\quad\quad\quad\quad \mathit{chart} \leftarrow \mathit{chart} \cup \mathit{agenda}[i] \\
\quad\quad\quad\quad \mathit{agenda}[i+1] \leftarrow \mathrm{Conseq}(\mathbf{P}, \mathit{agenda}[i], \mathit{chart}) - \mathit{chart} \\
\quad\quad\quad\quad i \leftarrow i + 1 \\
\textbf{return } \mathit{chart}
\end{array}
$$

holds facts with derivation tree of minimal height *i*

facts that immediately follow from one fact in *agenda*[*i*] plus some facts in *chart*

≈ well-formed substring table

Works for Datalog programs in general

Outputting shared forest

Seminaive_parse($\mathbf{P}$, D)

$agenda[0] \leftarrow D$
$i \leftarrow 0$
$chart \leftarrow \varnothing$
$parse \leftarrow \varnothing$
while $agenda[i] \neq \varnothing$
  do
    $chart \leftarrow chart \cup agenda[i]$
    $agenda[i+1] \leftarrow \mathrm{Conseq}(\mathbf{P}, agenda[i], chart) - chart$
    $parse \leftarrow parse \cup \mathrm{Inst}(\mathbf{P}, agenda[i], chart)$
    $i \leftarrow i + 1$
return $parse$

holds instances of rules that can be used in derivation trees

facts that immediately follow from one fact in *agenda*[*i*] plus some facts in *chart*

instances of rules with right-hand side consisting of one fact in *agenda*[i] and some facts in *chart*

shared parse forest

Parsing algorithms are often not explicitly stated in textbooks.

Computational complexity

- CFG

Fixed grammar recognition	LOGCFL-complete
Uniform recognition	P-complete
ε-free uniform recognition	LOGCFL-complete

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**

Fixed grammar recognition	LOGCFL-complete
Uniform recognition	P-complete
ε-free uniform recognition	LOGCFL-complete

width = $|\sigma(B)|$ = 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

$$\frac{[\mathrm{NP}, i, j] \quad [\mathrm{VP}, j, k]}{[\mathrm{S}, i, k]}$$

The idea of using Datalog is not new.
DCG is too powerful.
Datalog notation is more convenient.

CFG + Montague semantics

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

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

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

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

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

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

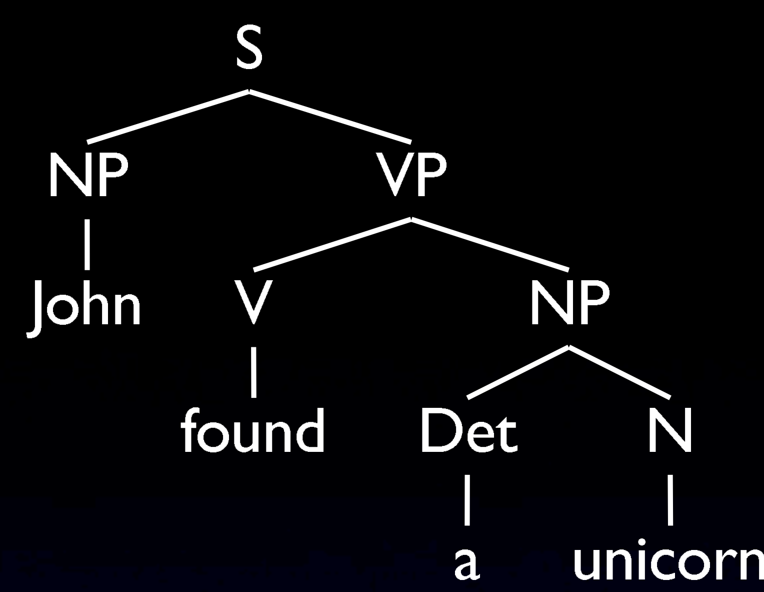
$$V(\mathbf{catch}^{e \rightarrow e \rightarrow t}) \rightarrow \text{caught}$$

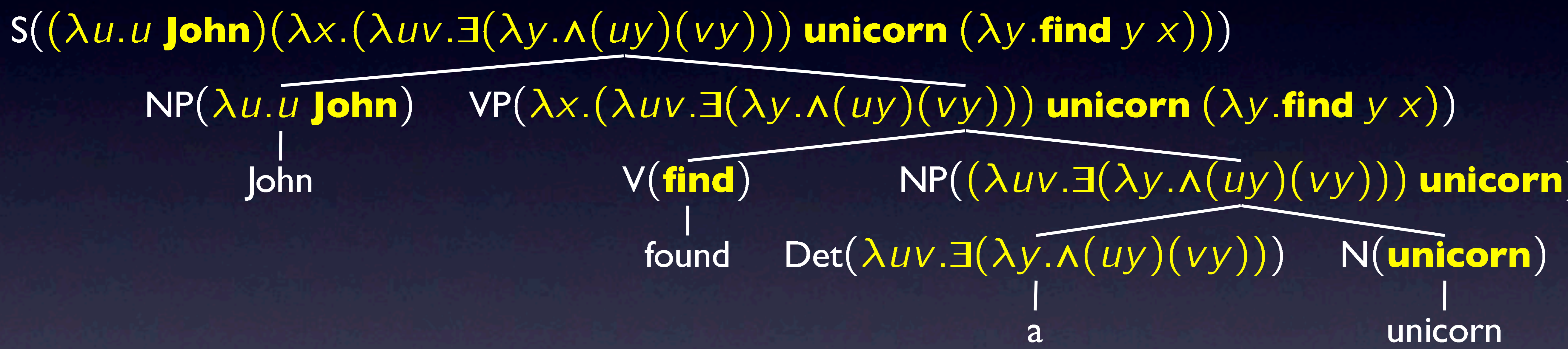
$$Conj(\wedge^{t \rightarrow t \rightarrow t}) \rightarrow \text{and} \qquad \lambda uv.\exists y(u(y) \wedge v(y))$$

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

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

One way of writing Montague semantics with CFG





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

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

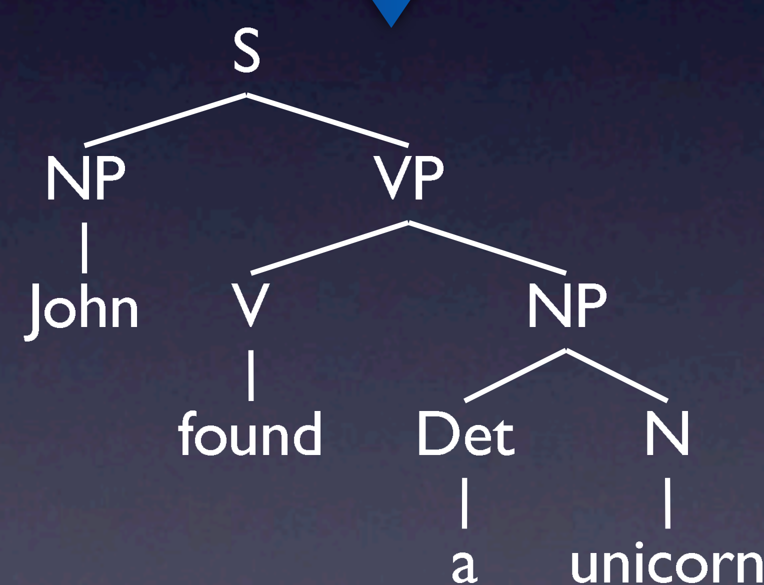
$$\approx\ \exists y(\mathbf{unicorn}(y) \wedge \mathbf{find}(\mathbf{John}, y))$$

logical form

Grammar rules associates a lambda-term to each node
Must reduce to normal form to get the desired representation

Surface realization

$\exists y(\mathbf{unicorn}(y) \wedge \mathbf{find}(\mathbf{John}, y))$ input logical form



derivation tree

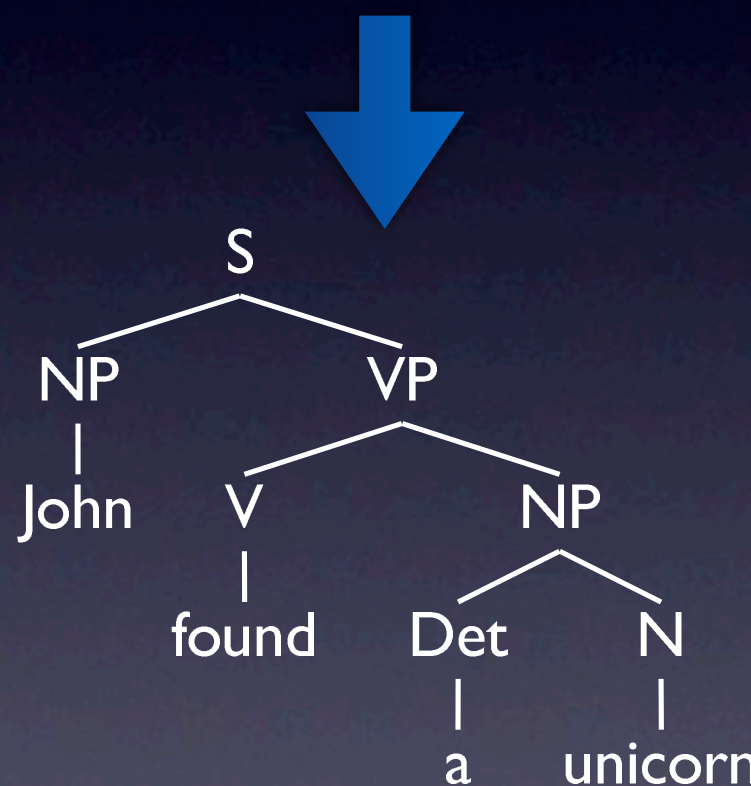
John found a unicorn output surface form

Tactical generation or surface realization

Parsing of input logical form

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

input logical form



derivation tree

Can concentrate on the semantic half of the grammar

Recognition of surface realizability

$\exists y(\mathbf{unicorn}(y) \wedge \mathbf{find}(\mathbf{John}, y))$ input logical form

surface realizable?

$\downarrow$

yes/no

Solving this problem almost amounts to solving surface realization

Context-free grammar on λ-terms

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

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

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

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

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

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

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

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

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

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

CFG + Montague - CFG = CFLG
Generates a set of lambda-terms

Context-free grammar on λ-terms

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

$$\sigma(\mathrm{S}) = t$$
$$\sigma(\mathrm{VP}) = \mathrm{e} \to t$$
$$\sigma(\mathrm{NP}) = (\mathrm{e} \to t) \to t$$
$$\sigma(\mathrm{V}) = \mathrm{e} \to \mathrm{e} \to t$$
$$\sigma(\mathrm{Conj}) = t \to t \to t$$
$$\sigma(\mathrm{Det}) = (\mathrm{e} \to t) \to (\mathrm{e} \to t) \to t$$
$$\sigma(\mathrm{N}) = \mathrm{e} \to t$$

A nonterminal is associated with a type.
Arguments are λ-terms of that type.

Context-free grammars on λ-terms = second-order non-linear ACGs

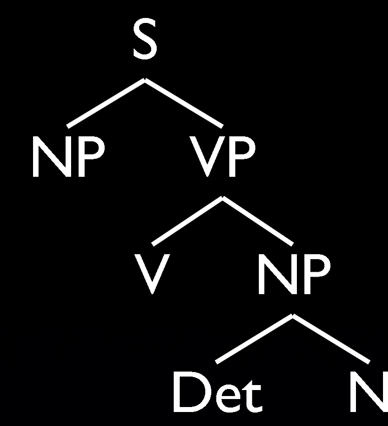
$$\pi\colon B(M) \coloneq B_1(X_1), \ldots, B_n(X_n).$$

$$\begin{array}{c} \pi : B_1 \to \cdots \to B_n \to B \\ \Big\downarrow \mathscr{L} \\ \lambda X_1 \ldots X_n . M \end{array}$$

type of π

abstract constant

object realization of π



$S(\lambda z.X_1(X_2 z)) \coloneq NP(X_1),\ VP(X_2)$

$VP(\lambda z.X_1(X_2 z)) \coloneq V(X_1), NP(X_2).$

$V(\lambda z.X_1(X_2(X_3 z))) \coloneq V(X_1), Conj(X_2), V(X_3).$

$NP(\lambda z.X_1(X_2 z)) \coloneq Det(X_1), N(X_2).$

$NP(\lambda z.\mathrm{John}\ z).$

$V(\lambda z.\mathrm{found}\ z).$

$V(\lambda z.\mathrm{caught}\ z).$

$Conj(\lambda z.\mathrm{and}\ z).$

$Det(\lambda z.\mathrm{a}\ z).$

$N(\lambda z.\mathrm{unicorn}\ z).$

$S(X_1 X_2) \coloneq NP(X_1),\ VP(X_2)$

$VP(\lambda x.X_2(\lambda y.X_1 yx)) \coloneq V(X_1), NP(X_2).$

$V(\lambda yx.X_2(X_1 yx)(X_3 yx)) \coloneq V(X_1), Conj(X_2), V(X_3).$

$NP(X_1 X_2) \coloneq Det(X_1), N(X_2).$

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

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

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

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

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

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

/John found a unicorn/

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

A pair of second-order (non-linear) ACGs as a “synchronous” grammar

From second-order ACG recognition to Datalog query evaluation

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

$VP(\lambda x.X_2(\lambda y.X_1 yx)) \coloneq V(X_1), NP(X_2).$

$V(\lambda yx.X_2(X_1 yx)(X_3 yx)) \coloneq V(X_1), Conj(X_2), V(X_3).$

$NP(X_1 X_2) \coloneq Det(X_1), N(X_2).$

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

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

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

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

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

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

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

From second-order ACG recognition to Datalog query evaluation

$S(i_1) \coloneq NP(i_1, i_2, i_3)\ VP(i_2, i_3)$

$VP(i_1, i_4) \coloneq V(i_2, i_4, i_3), NP(i_1, i_2, i_3).$

$V(i_1, i_4, i_3) \coloneq V(i_2, i_4, i_3), Conj(i_1, i_5, i_2), V(i_5, i_4, i_3).$

$NP(i_1, i_4, i_5) \coloneq Det(i_1, i_4, i_5, i_2, i_3), N(i_2, i_3).$

$NP(i_1, i_1, i_2) \coloneq \mathbf{John}(i_2).$

$V(i_1, i_3, i_2) \coloneq \mathbf{find}(i_1, i_3, i_2).$

$V(i_1, i_3, i_2) \coloneq \mathbf{catch}(i_1, i_3, i_2).$

$Conj(i_1, i_3, i_2) \coloneq \wedge(i_1, i_3, i_2).$

$Det(i_1, i_5, i_4, i_3, i_4) \coloneq \exists(i_1, i_2, i_4), \wedge(i_2, i_5, i_3).$

$N(i_1, i_2) \coloneq \mathbf{unicorn}(i_1, i_2).$

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

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

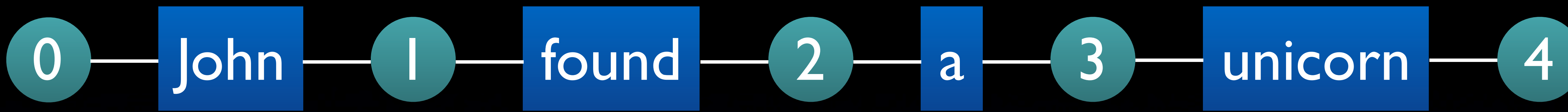
$\mathbf{unicorn}(3, 4).$

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

$\mathbf{John}(6).$

$?- S(1).$

Given conversion to Datalog, can use general Datalog techniques.

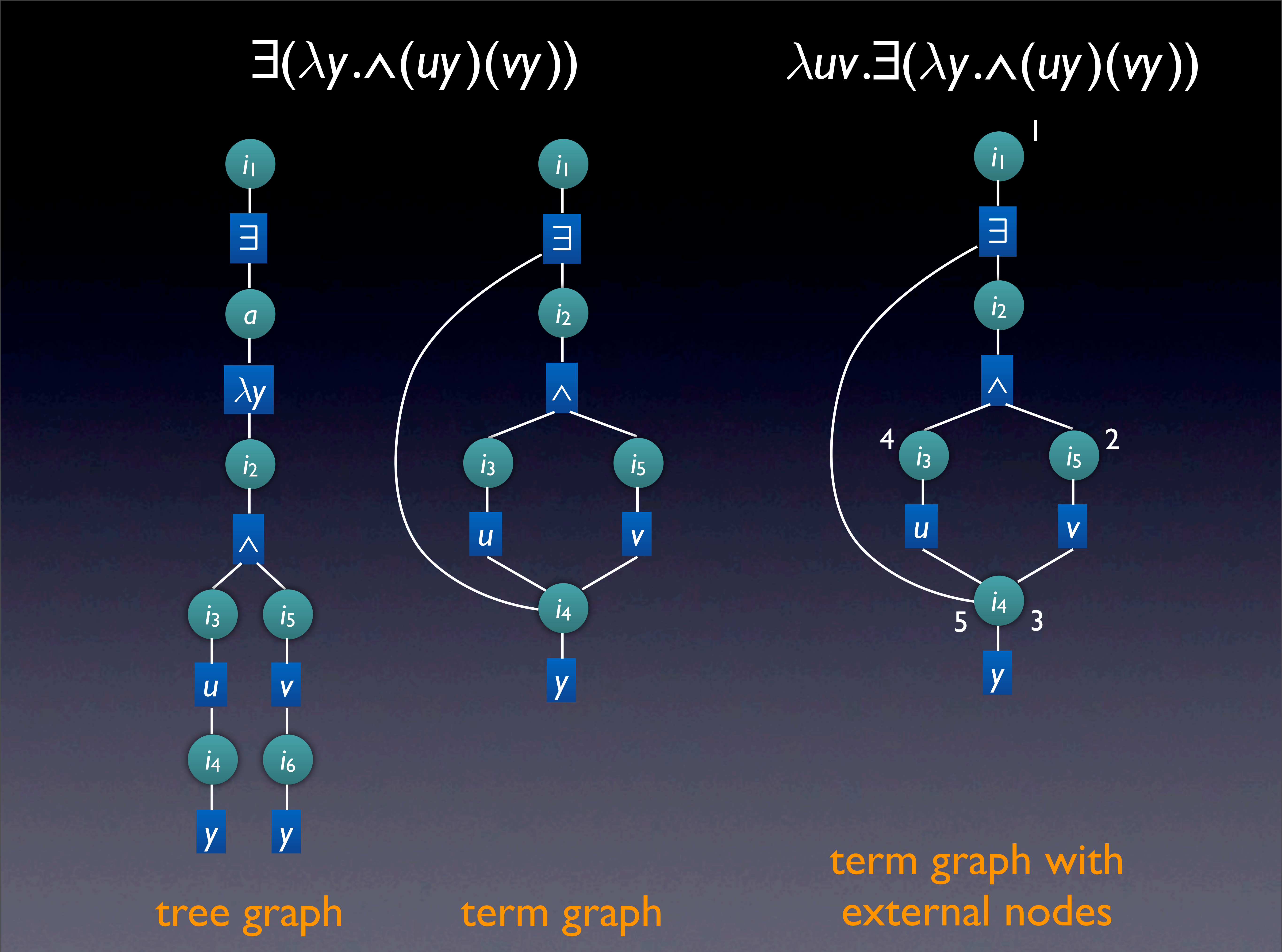


John(0, 1) found(1, 2) a(2, 3) unicorn(3, 4)

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

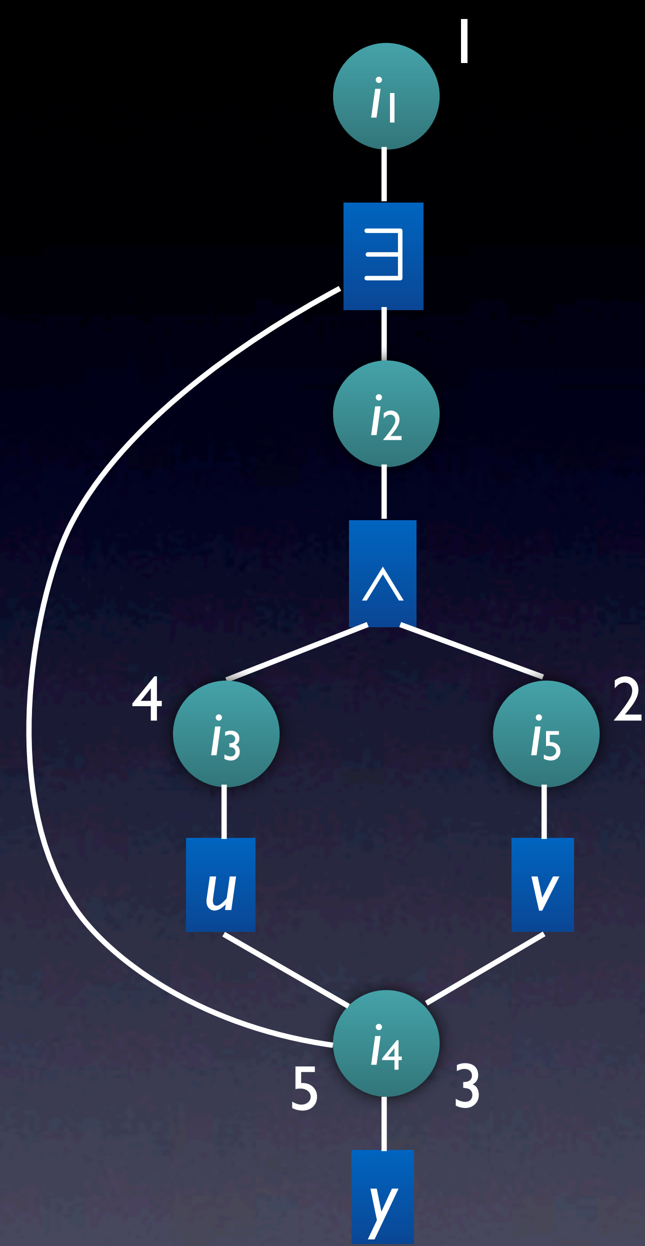
?- S(0, 4).

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.



λ–terms can also be represented by hypergraphs (when almost linear).
A hypergraph is a “term graph” when each node is the “result node” of a unique hyperedge.
A hyperedge is “directed”: the nodes it attaches to are ordered.

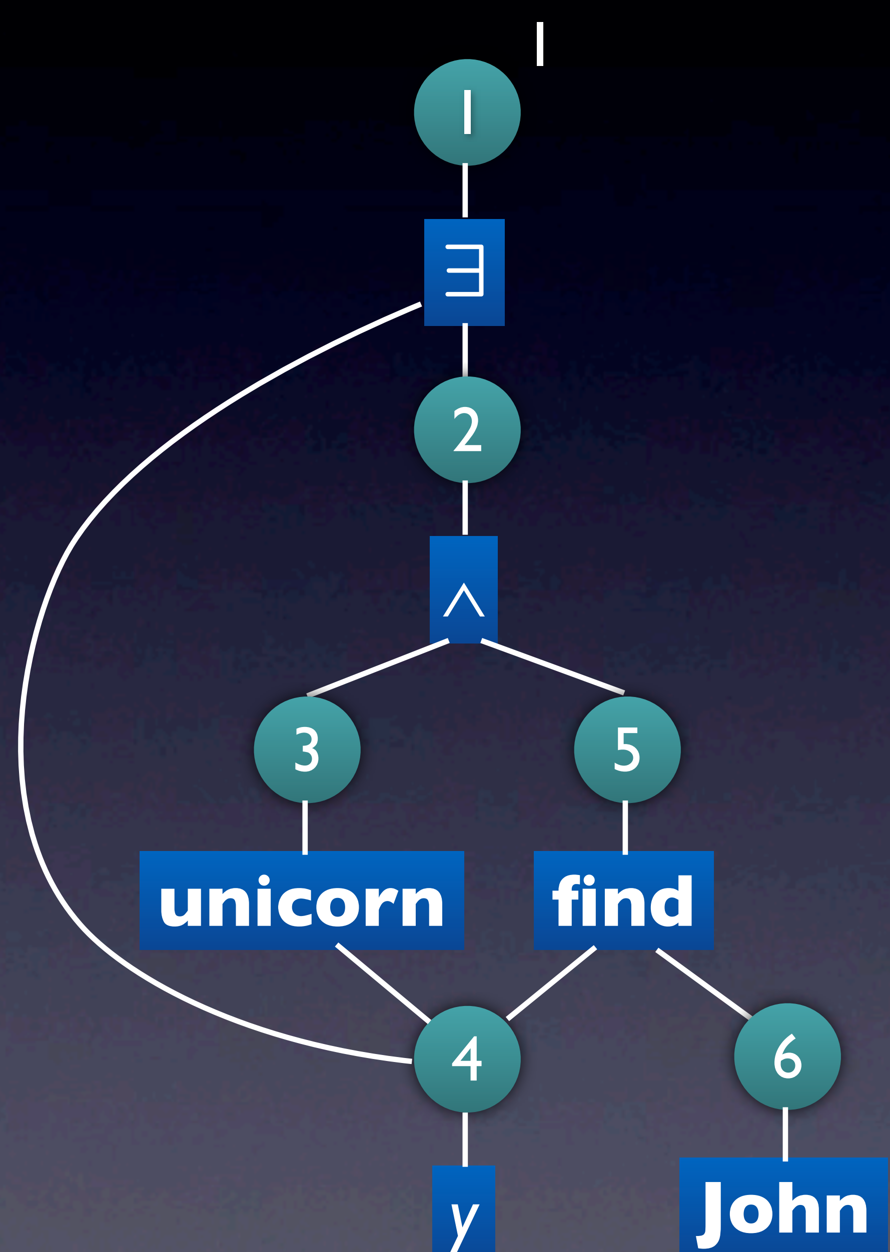
$$\mathrm{Det}(\lambda uv.\exists^{(e\to t)\to t}(\lambda y.\wedge^{t\to t\to t}(uy)(vy))).$$



$$\mathrm{Det}(i_1, i_5, i_4, i_3, i_4) :\!-\, \exists(i_1, i_2, i_4), \wedge(i_2, i_5, i_3).$$

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.

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



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

?– S(1).

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.

$$\frac{}{x : \alpha \vdash x : \alpha} \qquad \frac{}{\vdash c : \tau(c)}$$

$$\frac{\Gamma \vdash M : \alpha \quad \Delta \vdash N : \beta}{\Gamma \cup \Delta \vdash MN : \beta} \text{ if } \Gamma \cap \Delta \subseteq At \qquad \frac{\Gamma, x : \alpha \vdash M : \beta}{\Gamma \vdash \lambda x.M : \alpha \to \beta}$$

almost affine

λI

Vacuous abstraction is not allowed.
If a variable occurs twice in a subterm, it must have an atomic type.

Context-free grammar on λ-terms

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

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

$$V(\lambda y^e x^e.X_2(X_1 y^e x^e)(X_3 y^e x^e)) :- V(X_1), Conj(X_2), V(X_3).$$

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

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

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

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

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

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

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

All λ-terms almost linear.
Generates β-normal forms of almost linear λ-terms.

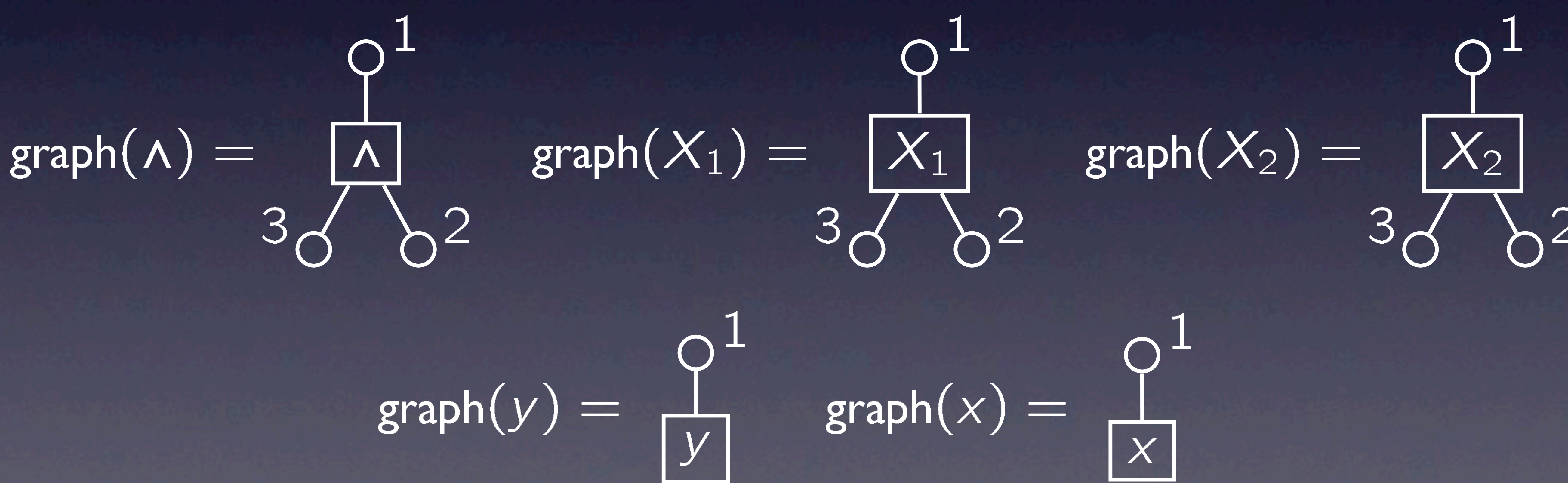
Almost linear λ-terms

- The class of almost linear λ-terms is not closed under β-reduction.

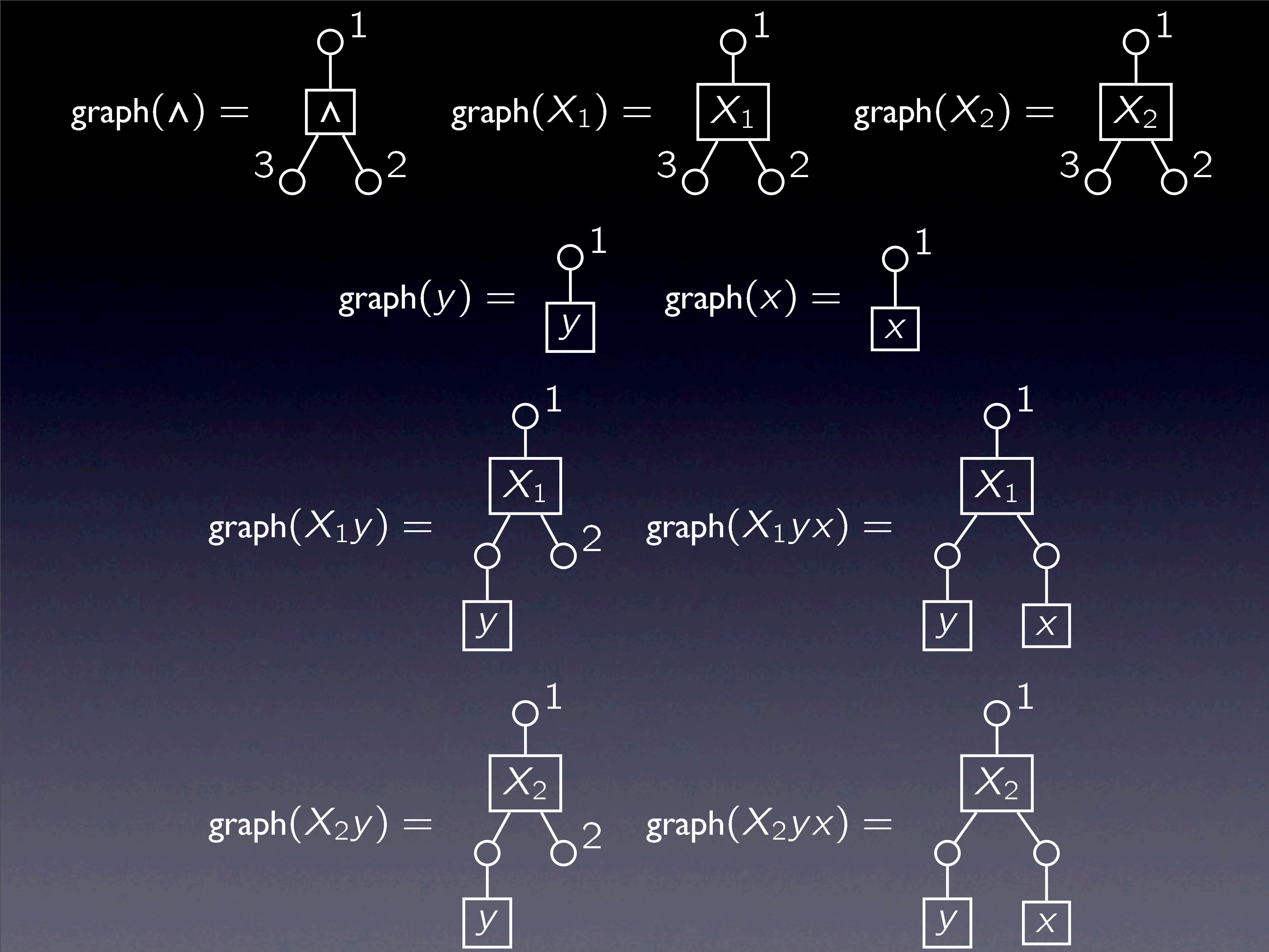
$$(\lambda x^{e}.y^{e\to e\to t}xx)(z^{e\to e}w^{e}) \to_{\beta} y(zw)(zw)$$

A formal definition of graph(M)

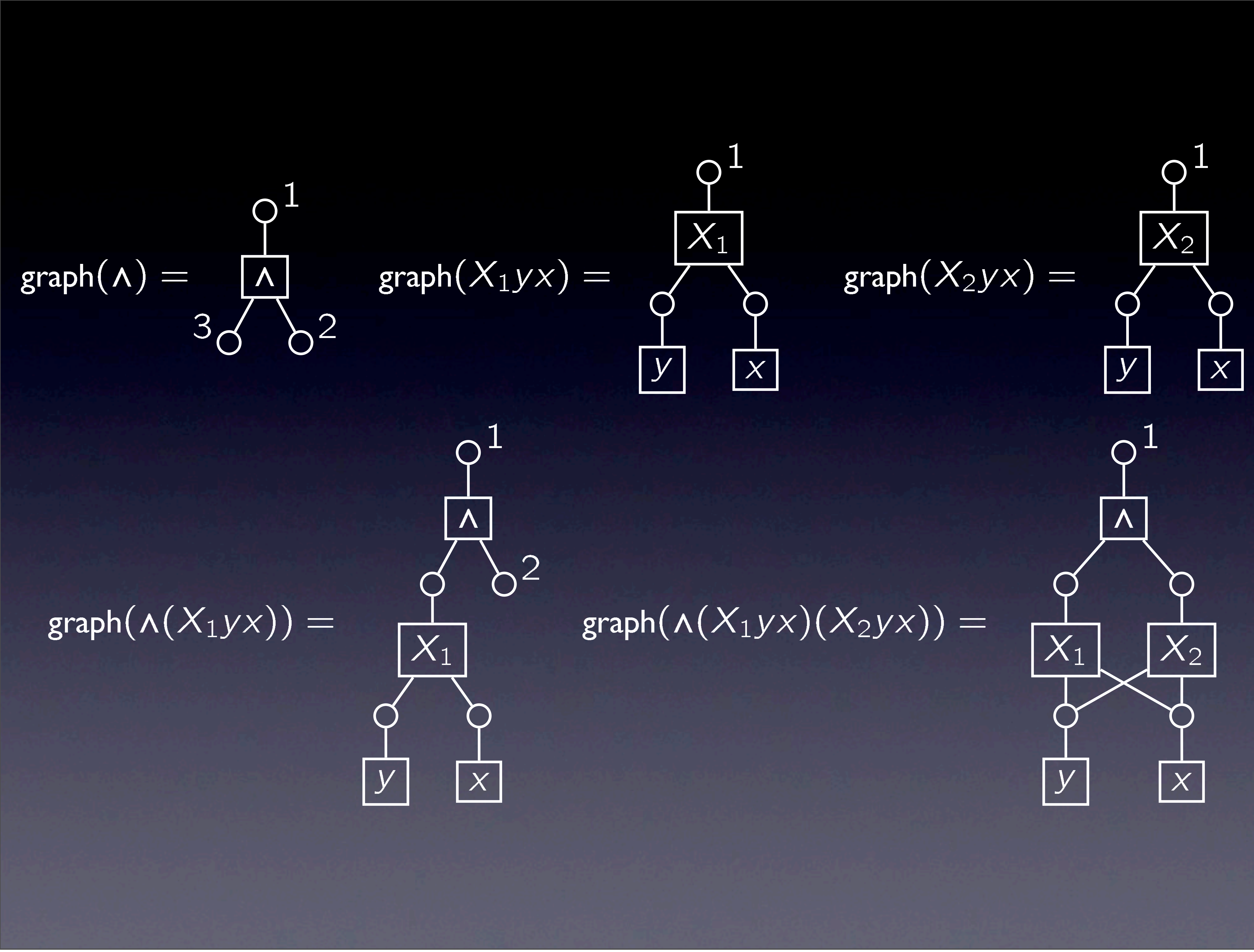
$$\lambda y^{e} x^{e} . \wedge^{t\to t\to t}(X_{1}^{e\to e\to t} y x)(X_{3} y x)$$



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.

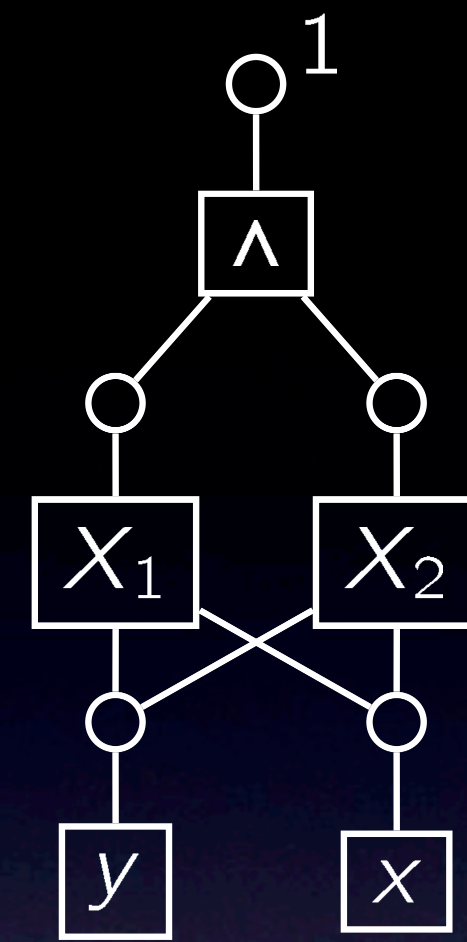


For $M^{\alpha\to\beta}N^{\alpha}$, the last $|\alpha|$ external nodes of graph(M) are identified with the external nodes of graph(N).
The new external nodes are the remaining external nodes of graph(M).

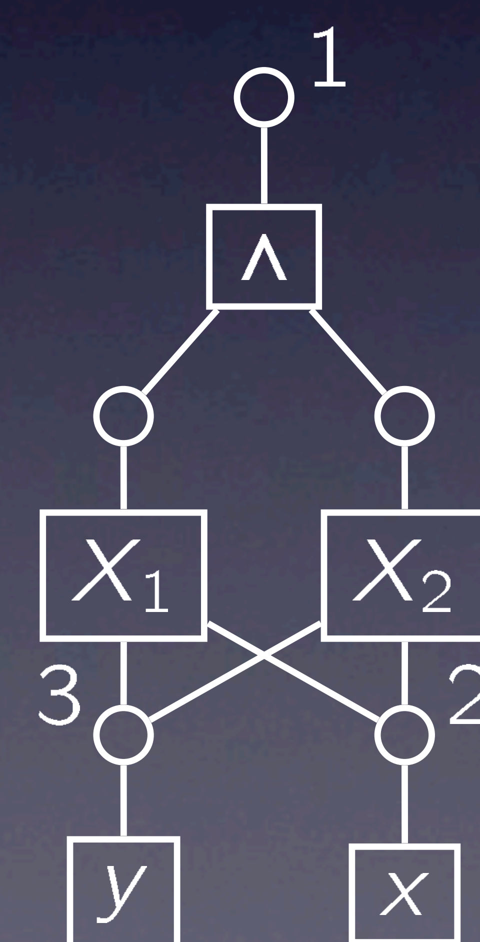


The edges labeled by the same variable (and the nodes they attach to) are also merged.
Such a variable is atomic-typed.

$$\mathrm{graph}(\wedge(X_1 yx)(X_2 yx)) =$$

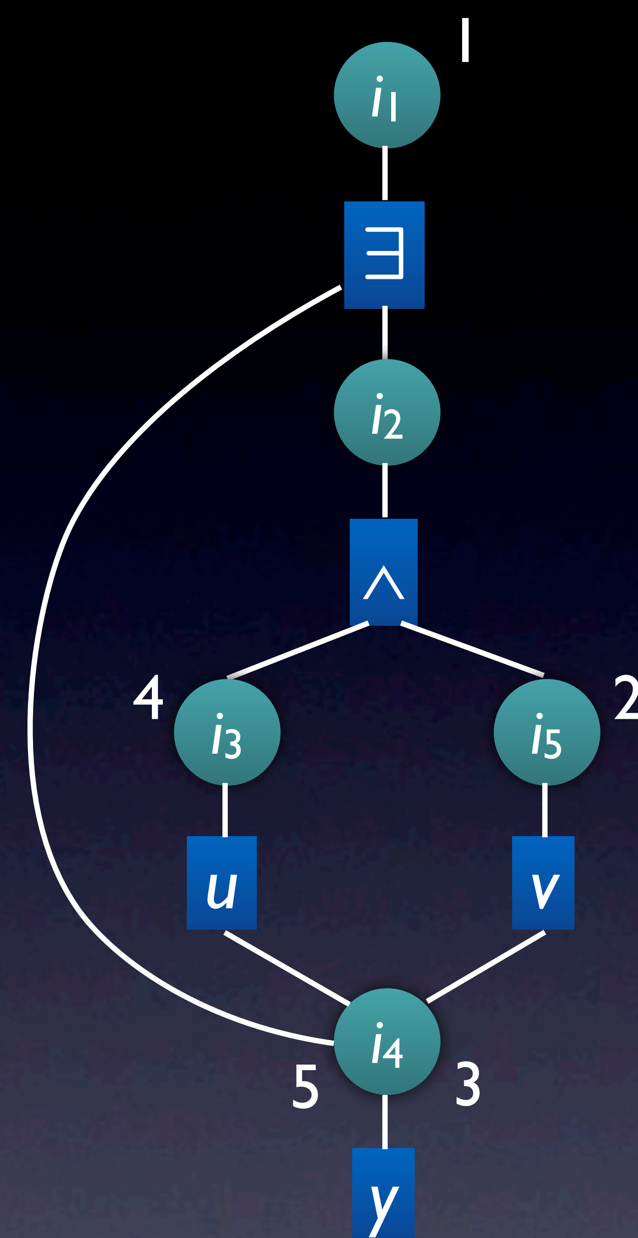


$$\mathrm{graph}(\lambda yx.\wedge(X_1 yx)(X_2 yx)) =$$



The nodes that the abstracted variable attaches to are appended to the list of external nodes.

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



$$\exists : (e \to t) \to t, \wedge : t \to t \to t \vdash M : (e \to t) \to (e \to t) \to t$$

$$\exists : (i_4 \to i_2) \to i_1, \wedge : i_3 \to i_5 \to i_2 \vdash M : (i_4 \to i_3) \to (i_4 \to i_5) \to i_1$$

principal typing

The construction of the graph gives a principal typing (i.e., most general typing).

Typed λ-calculus

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

$$M \twoheadrightarrow_\beta M' \text{ and } \Gamma \vdash M : \alpha \Longrightarrow \Gamma \vdash M' : \alpha$$

General facts of importance.
All other typings are instantiations of the principal typing.

Pure linear λ-terms

- The principal typing of an affine λ-term is balanced.

  Belnap 1976

- If M has a balanced typing, it is affine.

  Hirokawa 1992

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

Many properties of linear λ–terms carry over to almost linear.
Linear = affine + λI
"Balanced" means that there is at most one positive and at most one negative occurrence of any atomic type.

Pure linear λ-terms

- **Coherence Theorem.** All inhabitants of a balanced typing are βη-equal.

Babaev and Solov'ev 1979

Pure linear λ-terms

- **Subject Expansion Theorem.**

$M \twoheadrightarrow_\beta M'$ and $\Gamma \vdash M' : \alpha \Longrightarrow \Gamma \vdash M : \alpha$

non-erasing
non-duplicating

Hindley

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

$$\downarrow_\beta$$

$$x : (i \to i) \to (j \to j) \to k \vdash x(\lambda z.z)(\lambda w.w) : k$$

Pure almost linear λ-terms

- The principal typing of an almost affine λ-term is negatively non-duplicated.

  Aoto 1999

- If M has a negatively non-duplicated typing, it is βη-equal to an almost affine λ-term.

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

$$\exists : (\overset{-}{i_4} \to \overset{+}{i_2}) \to \overset{-}{i_1},\ \wedge : \overset{+}{i_3} \to \overset{+}{i_5} \to \overset{-}{i_2} \vdash M : (\overset{+}{i_4} \to \overset{-}{i_3}) \to (\overset{+}{i_4} \to \overset{-}{i_5}) \to \overset{+}{i_1}$$

Almost linear = almost affine + λI
"Negatively non-duplicated" means that there is at most one negative occurrence of any atomic type.

Pure almost linear λ-terms

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

Aoto and Ono 1994

Pure almost linear λ-terms

- **Subject Expansion Theorem.**

$M \twoheadrightarrow_\beta M'$ and $\Gamma \vdash M' : \alpha \Longrightarrow \Gamma \vdash M : \alpha$

non-erasing

almost non-duplicating

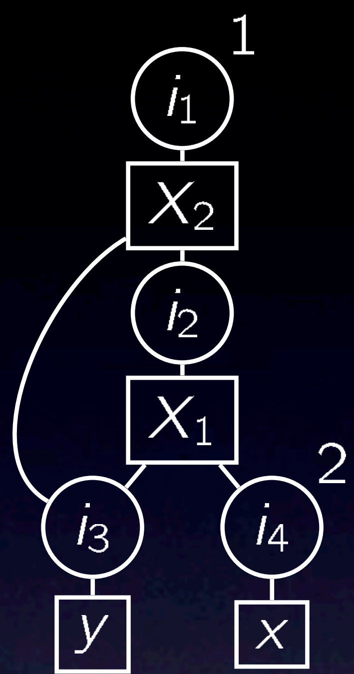
- The leftmost reduction from an almost affine λ-term is almost non-duplicating.

$$(\lambda w.(\lambda x.yxx)(wz))(\lambda v.v)$$

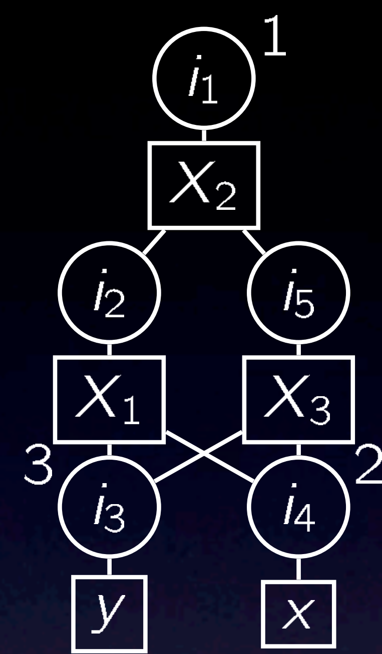
$$(\lambda x.yxx)((\lambda v.v)z) \xleftarrow{\beta} (\lambda w.(\lambda x.yxx)(wz))(\lambda v.v) \xrightarrow{\beta} (\lambda w.y(wz)(wz))(\lambda v.v)$$

A reduction is “almost non-duplicating” if any duplicating contraction involves λ binding an atomic typed variable.

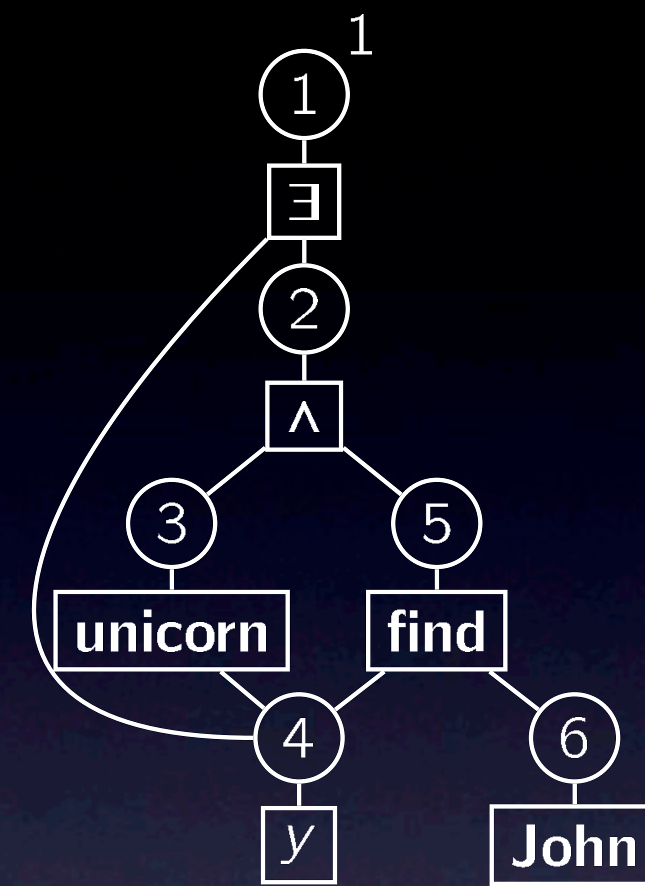
$\mathsf{VP}(\lambda x.X_2(\lambda y.X_1 yx))$:–
$\mathsf{V}(X_1), \mathsf{NP}(X_2).$



$\mathsf{V}(\lambda yx.X_2(X_1 yx)(X_3 yx))$:–
$\mathsf{V}(X_1), \mathsf{Conj}(X_2), \mathsf{V}(X_3).$



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



$\mathsf{S}(i_1)$:– $\mathsf{NP}(i_1, i_2, i_3), \mathsf{VP}(i_2, i_3).$
$\mathsf{VP}(i_1, i_4)$:– $\mathsf{V}(i_2, i_4, i_3), \mathsf{NP}(i_1, i_2, i_3).$
$\mathsf{V}(i_1, i_4, i_3)$:– $\mathsf{V}(i_2, i_4, i_3), \mathsf{Conj}(i_1, i_5, i_2), \mathsf{V}(i_5, i_4, i_3).$
$\mathsf{NP}(i_1, i_4, i_5)$:– $\mathsf{Det}(i_1, i_4, i_5, i_2, i_3), \mathsf{N}(i_2, i_3).$
$\mathsf{NP}(i_1, i_1, i_2)$:– $\textbf{John}(i_2).$
$\mathsf{V}(i_1, i_3, i_2)$:– $\textbf{find}(i_1, i_3, i_2).$
$\mathsf{V}(i_1, i_3, i_2)$:– $\textbf{catch}(i_1, i_3, i_2).$
$\mathsf{Conj}(i_1, i_3, i_2)$:– $\wedge(i_1, i_3, i_2).$
$\mathsf{Det}(i_1, i_5, i_4, i_3, i_4)$:– $\exists(i_1, i_2, i_4), \wedge(i_2, i_5, i_3).$
$\mathsf{N}(i_1, i_2)$:– $\textbf{unicorn}(i_1, i_2).$

Datalog program

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

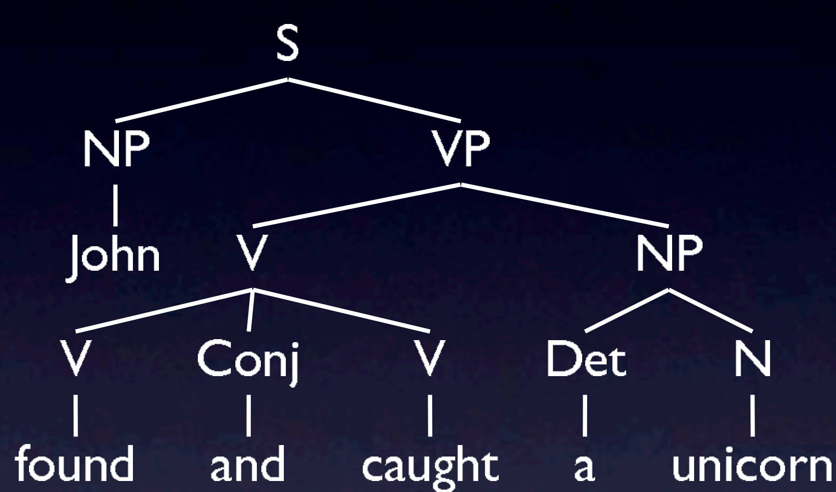
database

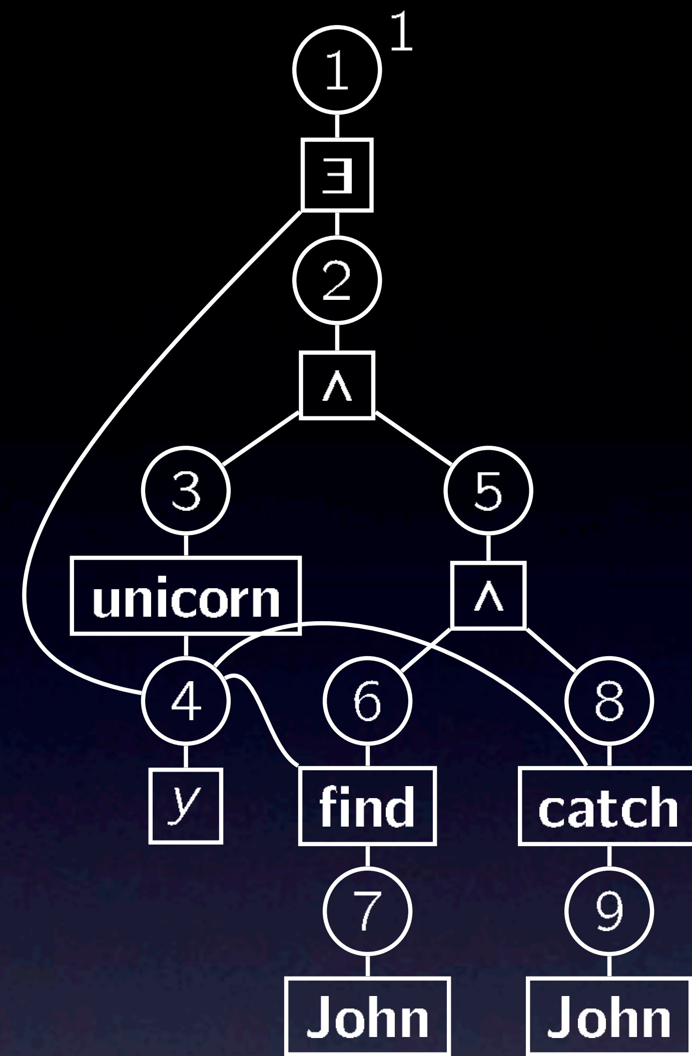
?– $\mathsf{S}(1).$

query

yes!

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





$\mathrm{S}(i_1) \coloneq \mathrm{NP}(i_1, i_2, i_3), \mathrm{VP}(i_2, i_3).$
$\mathrm{VP}(i_1, i_4) \coloneq \mathrm{V}(i_2, i_4, i_3), \mathrm{NP}(i_1, i_2, i_3).$
$\mathrm{V}(i_1, i_4, i_3) \coloneq \mathrm{V}(i_2, i_4, i_3), \mathrm{Conj}(i_1, i_5, i_2), \mathrm{V}(i_5, i_4, i_3).$
$\mathrm{NP}(i_1, i_4, i_5) \coloneq \mathrm{Det}(i_1, i_4, i_5, i_2, i_3), \mathrm{N}(i_2, i_3).$
$\mathrm{NP}(i_1, i_1, i_2) \coloneq \textbf{John}(i_2).$
$\mathrm{V}(i_1, i_3, i_2) \coloneq \textbf{find}(i_1, i_3, i_2).$
$\mathrm{V}(i_1, i_3, i_2) \coloneq \textbf{catch}(i_1, i_3, i_2).$
$\mathrm{Conj}(i_1, i_3, i_2) \coloneq \wedge(i_1, i_3, i_2).$
$\mathrm{Det}(i_1, i_5, i_4, i_3, i_4) \coloneq \exists(i_1, i_2, i_4), \wedge(i_2, i_5, i_3).$
$\mathrm{N}(i_1, i_2) \coloneq \textbf{unicorn}(i_1, i_2).$

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

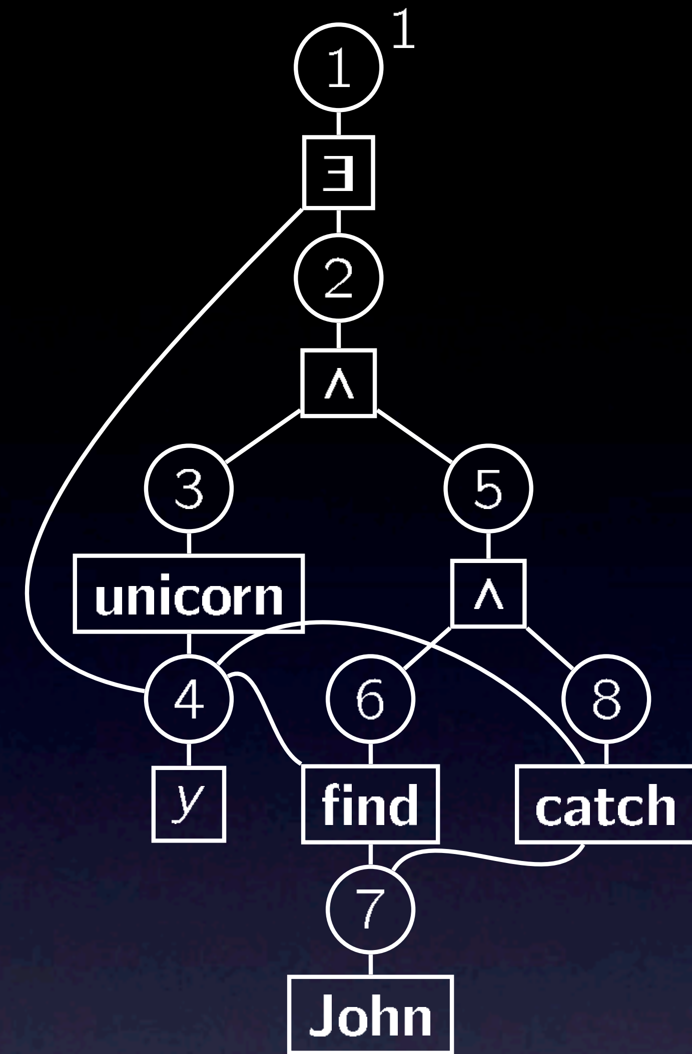
$?\!-\ \mathrm{S}(1).$

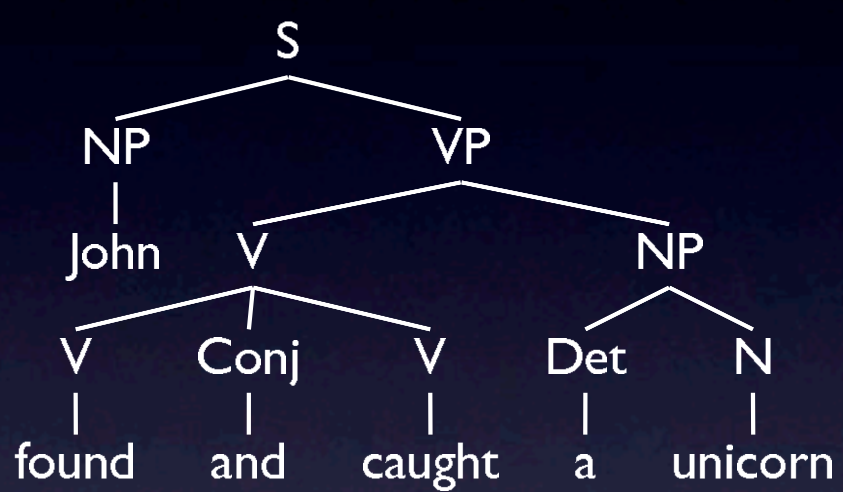
query

no!

A tricky case.

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





$\mathsf{S}(i_1) \coloneq \mathsf{NP}(i_1, i_2, i_3), \mathsf{VP}(i_2, i_3).$
$\mathsf{VP}(i_1, i_4) \coloneq \mathsf{V}(i_2, i_4, i_3), \mathsf{NP}(i_1, i_2, i_3).$
$\mathsf{V}(i_1, i_4, i_3) \coloneq \mathsf{V}(i_2, i_4, i_3), \mathsf{Conj}(i_1, i_5, i_2), \mathsf{V}(i_5, i_4, i_3).$
$\mathsf{NP}(i_1, i_4, i_5) \coloneq \mathsf{Det}(i_1, i_4, i_5, i_2, i_3), \mathsf{N}(i_2, i_3).$
$\mathsf{NP}(i_1, i_1, i_2) \coloneq \textbf{John}(i_2).$
$\mathsf{V}(i_1, i_3, i_2) \coloneq \textbf{find}(i_1, i_3, i_2).$
$\mathsf{V}(i_1, i_3, i_2) \coloneq \textbf{catch}(i_1, i_3, i_2).$
$\mathsf{Conj}(i_1, i_3, i_2) \coloneq \wedge(i_1, i_3, i_2).$
$\mathsf{Det}(i_1, i_5, i_4, i_3, i_4) \coloneq \exists(i_1, i_2, i_4), \wedge(i_2, i_5, i_3).$
$\mathsf{N}(i_1, i_2) \coloneq \textbf{unicorn}(i_1, i_2).$

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

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

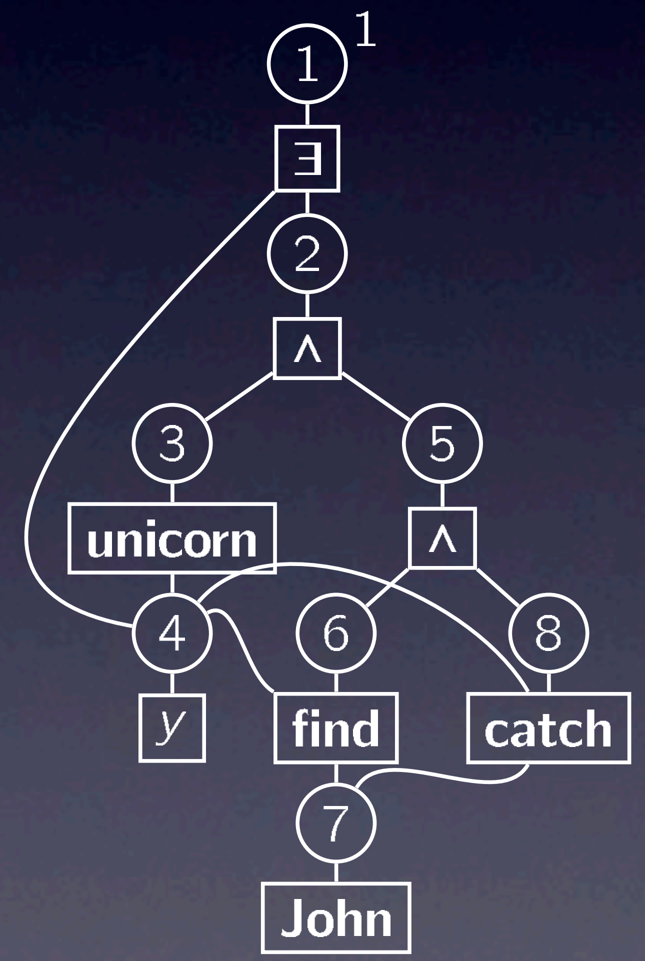
query

yes!

Reduction to Datalog

- Given an input λ-term, find the most compact term graph (≈ fully collapsed form) representing it.

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



- This graph represents a pure almost linear λ-term.

$$M' = (\lambda x.\exists(\lambda y.\wedge_1(\textbf{unicorn}\ y)(\wedge_2(\textbf{find}\ y\ x)(\textbf{catch}\ y\ x))))\ \textbf{John}$$

The two occurrences of John are identified, but not the two occurrences of ∧.

Reduction to Datalog

$$\begin{array}{l} \exists(1,2,4). \\ \wedge(2,5,3). \\ \textbf{unicorn}(3,4). \\ \wedge(5,8,6). \\ \textbf{find}(6,7,4). \\ \textbf{John}(7). \\ \textbf{catch}(8,7,4). \end{array} \qquad ?- S(1).$$

- (database(M'), query(M')) represents a set of λ-terms.

$$\mathcal{L} = \{ N' \mid \exists : (4 \to 2) \to 1, \wedge_1 : 3 \to 5 \to 2, \textbf{unicorn} : 4 \to 3,$$
$$\wedge_2 : 6 \to 8 \to 5, \textbf{find} : 6 \to 7 \to 5, \textbf{John} : 7 \vdash N' : 1 \}$$

- All elements of $\mathcal{L}$ are βη-equal by Aoto and Ono's Coherence Theorem.
- This set contains all almost linear λ-terms that β-reduce to $|M'|_\beta$ by the Subject Expansion Theorem.
- But this is not enough!

Reduction to Datalog

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

$?- S(1).$

- (database(M'), query(M')) represents a set of λ-terms.

$$\mathcal{L} = \{ N' \mid \exists : (4 \to 2) \to 1, \wedge_1 : 3 \to 5 \to 2, \textbf{unicorn} : 4 \to 3,$$
$$\wedge_2 : 6 \to 8 \to 5, \textbf{find} : 6 \to 7 \to 5, \textbf{John} : 7 \vdash N' : 1 \}$$

- Since M' is the most compact almost linear λ-term such that $M'\theta \twoheadrightarrow_\beta M$ (where θ is the substitution that gives back the original constants), for every almost linear N such that $N \twoheadrightarrow_\beta M$, there is an $N' \in \mathcal{L}$ such that $N'\theta = N$.

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

$$\begin{tikzcd}
N[\wedge, \wedge, \mathbf{John}] \arrow[r, twoheadrightarrow] & M[\wedge, \wedge, \mathbf{John}, \mathbf{John}] \\
N'[\wedge_1, \wedge_2, \mathbf{John}] \arrow[u, "\theta"] \arrow[r, twoheadrightarrow] & M[\wedge_1, \wedge_2, \mathbf{John}, \mathbf{John}] \arrow[u, "\theta"] \\
& M'[\wedge_1, \wedge_2, \mathbf{John}] \arrow[u, twoheadrightarrow]
\end{tikzcd}$$

$$(\lambda x.\exists(\lambda y.\wedge_1(\mathbf{unicorn}\ y)(\wedge_2(\mathbf{find}\ y\ x)(\mathbf{catch}\ y\ x))))\ \mathbf{John}$$

$$\mathrm{S}(1)$$

$$\mathrm{NP}(1,1,7) \quad \mathrm{VP}(1,7)$$

$$\textbf{John}(7) \quad \mathrm{V}(5,7,4) \quad \mathrm{NP}(1,5,4)$$

$$\mathrm{V}(6,7,4) \quad \mathrm{Conj}(5,8,6) \quad \mathrm{V}(8,7,4) \quad \mathrm{Det}(1,5,4,3,4) \quad \mathrm{N}(3,4)$$

$$\textbf{find}(6,7,4) \quad \wedge(5,8,6) \quad \textbf{catch}(8,9,4) \quad \exists(1,2,4) \quad \wedge(2,5,3) \quad \textbf{unicorn}(3,4)$$

$$\Downarrow$$

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

$$\mathrm{NP}(\lambda u.u\ \textbf{John}) \quad \mathrm{VP}(\lambda x.(\lambda uv.\exists(\lambda y.\wedge(uy)(vy))\ \textbf{unicorn}\ (\lambda y.(\lambda yx.\wedge(\textbf{find}\ y\ x)(\textbf{catch}\ y\ x))\ y\ x)))$$

$$\mathrm{V}(\lambda yx.\wedge(\textbf{find}\ y\ x)(\textbf{catch}\ y\ x)) \quad \mathrm{NP}(\lambda uv.\exists(\lambda y.\wedge(uy)(vy))\ \textbf{unicorn})$$

$$\mathrm{V}(\textbf{find}) \quad \mathrm{Conj}(\wedge) \quad \mathrm{V}(\textbf{catch}) \quad \mathrm{Det}(\lambda uv.\exists(\lambda y.\wedge(uy)(\lambda y.vy))) \quad \mathrm{N}(\textbf{unicorn})$$

$$\textbf{John} : 7, \textbf{find} : 4 \to 7 \to 6, \wedge_2 : 6 \to 8 \to 5, \textbf{catch} : 4 \to 9 \to 8, \exists : (4 \to 2) \to 1, \wedge_1 : 3 \to 5 \to 2, \textbf{unicorn} : 4 \to 3$$
$$\vdash (\lambda u.u\ \textbf{John})(\lambda x.(\lambda uv.\exists(\lambda y.\wedge_1(uy)(vy))\ \textbf{unicorn}\ (\lambda y.(\lambda yx.\wedge_2\ (\textbf{find}\ y\ x)(\textbf{catch}\ y\ x))\ y\ x))) : 1$$

Aoto and Ono's Coherence Theorem $\Downarrow$

$$(\lambda u.u\ \textbf{John})(\lambda x.(\lambda uv.\exists(\lambda y.\wedge_1(uy)(vy))\ \textbf{unicorn}\ (\lambda y.(\lambda yx.\wedge_2(\textbf{find}\ y\ x)(\textbf{catch}\ y\ x))\ y\ x)))$$
$$\twoheadrightarrow_\beta \exists(\lambda y.\wedge_1(\textbf{unicorn}\ x)(\wedge_2(\textbf{find}\ y\ \textbf{John})(\textbf{catch}\ y\ \textbf{John}))$$

$$\Downarrow$$

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

A Datalog derivation tree determines a grammar derivation plus a typing of the associated λ-term.

S(1)

NP(1, 1, 7) — **John**(7)

VP(1, 7)

V(5, 7, 4)   NP(1, 5, 4)

V(6, 7, 4)   Conj(5, 8, 6)   V(8, 7, 4)   Det(1, 5, 4, 3, 4)   N(3, 4)

find(6, 7, 4) $\wedge$(5, 8, 6)   **catch**(8, 9, 4) $\exists$(1, 2, 4) $\wedge$(2, 5, 3)   **unicorn**(3, 4)

$\Uparrow$

$$\textbf{John} : 7, \textbf{find} : 4 \to 7 \to 6, \wedge_2 : 6 \to 8 \to 5, \textbf{catch} : 4 \to 9 \to 8, \exists : (4 \to 2) \to 1, \wedge_1 : 3 \to 5 \to 2, \textbf{unicorn} : 4 \to 3$$
$$\vdash (\lambda u.u\ \textbf{John})(\lambda x.(\lambda uv.\exists(\lambda y.\wedge_1(uy)(vy))\ \textbf{unicorn}\ (\lambda y.(\lambda yx.\wedge_2(\textbf{find}\ y\ x)(\textbf{catch}\ y\ x))\ y\ x))) : 1$$

Subject Expansion $\Uparrow$

S(($\lambda u.u$ **John**)($\lambda x.(\lambda uv.\exists(\lambda y.\wedge(uy)(vy))$ **unicorn** ($\lambda y.(\lambda yx.\wedge($**find** $y\ x)($**catch** $y\ x))\ y\ x))))

NP($\lambda u.u$ **John**)   VP($\lambda x.(\lambda uv.\exists(\lambda y.\wedge(uy)(vy))$ **unicorn** ($\lambda y.(\lambda yx.\wedge($**find** $y\ x)($**catch** $y\ x))\ y\ x)))

V($\lambda yx.\wedge($**find** $y\ x)($**catch** $y\ x)$)   NP($\lambda uv.\exists(\lambda y.\wedge(uy)(vy))$ **unicorn**)

V(**find**)   Conj($\wedge$)   V(**catch**)   Det($\lambda uv.\exists(\lambda y.\wedge(uy)(\lambda y.vy))$)   N(**unicorn**)

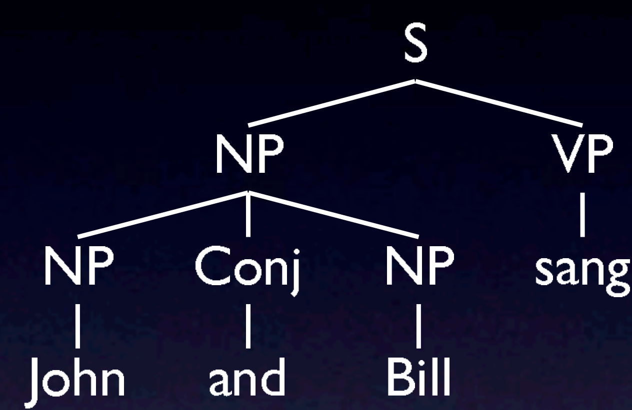
$$(\lambda u.u\ \textbf{John})(\lambda x.(\lambda uv.\exists(\lambda y.\wedge_1(uy)(vy))\ \textbf{unicorn}\ (\lambda y.(\lambda yx.\wedge_2(\textbf{find}\ y\ x)(\textbf{catch}\ y\ x))\ y\ x)))$$
$$\twoheadrightarrow_\beta \exists(\lambda y.\wedge_1(\textbf{unicorn}\ x)(\wedge_2(\textbf{find}\ y\ \textbf{John})(\textbf{catch}\ y\ \textbf{John}))$$

$\Uparrow$

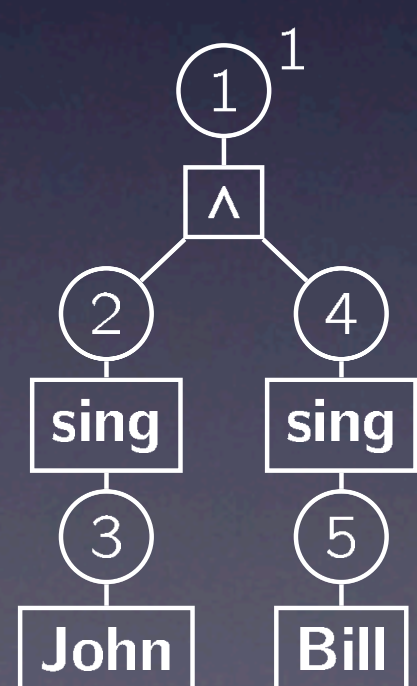
$$\exists(\lambda y.\wedge(\textbf{unicorn}\ x)(\wedge(\textbf{find}\ y\ \textbf{John})(\textbf{catch}\ y\ \textbf{John})) \in L(G)$$

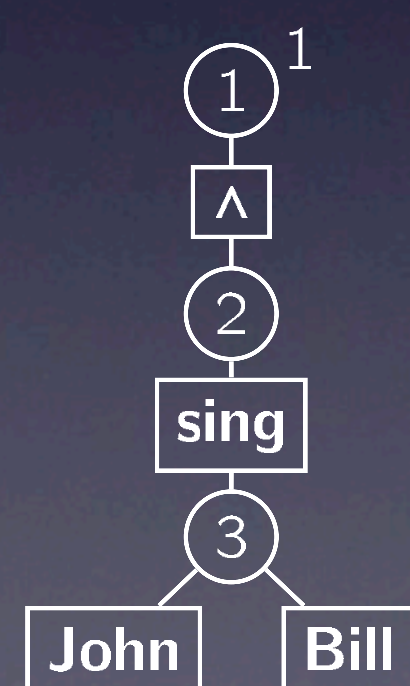
Limitations

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



$\wedge$(**sing John**)(**sing Bill**)

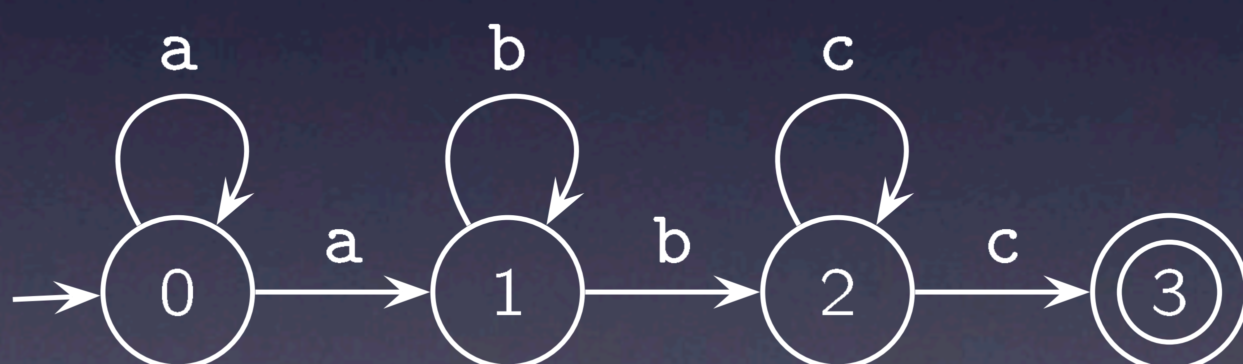




not a term graph

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*).




```
a(0, 0).   a(0, 1).
b(1, 1).   b(1, 2).
c(2, 2).   c(2, 3).
```

database

```
?– S(0, 3).
```

query

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