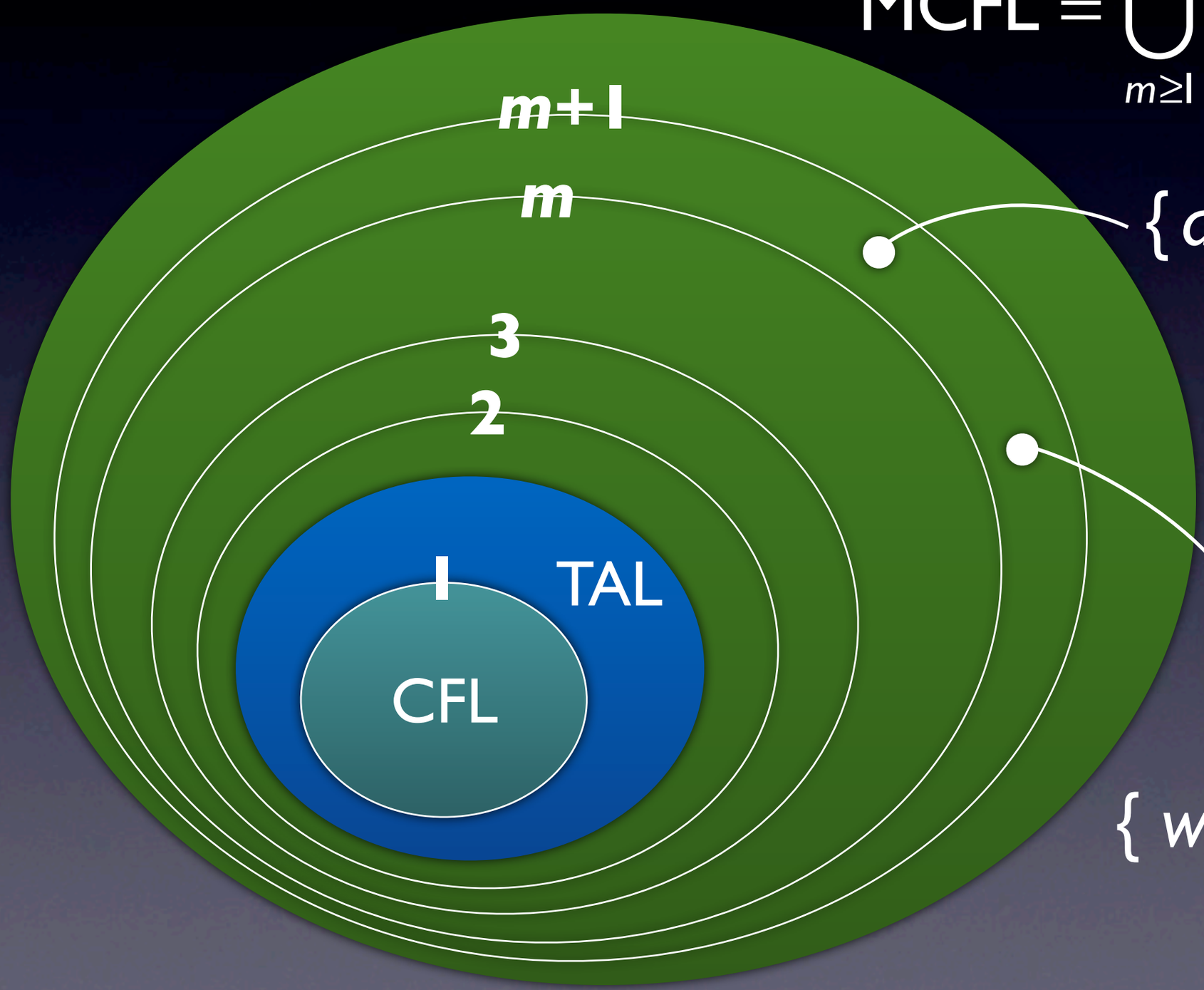
MCFL Dimension Hierarchy

$$\text{MCFL} = \bigcup_{m \geq 1} m\text{-MCFL}$$



$$\{ a_1^n \ldots a_{2m+1}^n \mid n \geq 0 \}$$

$$\{ w^{m+1} \mid w \in \{a, b\}^* \}$$

Seki et al.'s weak pumping lemma suffices to show that the hierarchy is infinite.

Theorem (Seki et al. 1991).
$L \in m$-MCFL $\Rightarrow L$ is weakly $2m$-iterative.

Myth. $L \in m$-MCFL $\Rightarrow L$ is $2m$-iterative.

Radzinski 1991, Groenink 1997, Kracht 2003

L is weakly k-iterative if it contains an infinite subset that is k-iterative.
Many people erroneously believed that Seki et al. proved a stronger result.

Coordination + Cross-Serial Dependencies

. . . dat	Jan	Piet	Marie	liet	opbellen,	hoorde uitnodigen,
that				*made*	*call*	*heard* *invite*

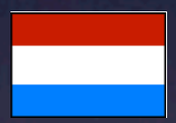
hielp	ontmoeten	en zag	omhelzen
helped	*meet*	*saw*	*embrace*

. . . that Jan made Piet call Marie, heard [him] invite [her], helped [him] meet [her] and saw [him] embrace [her]

dat Jan Piet Marie Fred^{n} (hoorde leren^{n} uitnodigen,)$^{k-1}$ en zag leren^{n} omhelzen

$$L \in m\text{-MCFL} \wedge R \in \text{REG} \Rightarrow L \cap R \in m\text{-MCFL}$$

$$L \in m\text{-MCFL} \Rightarrow h(L) \in m\text{-MCFL}$$

$h($  $\cap R) = \{ (a\, b^n)^m \mid m, n \geq 1 \}$

Groenink 1997

m-MCFL is closed under familiar operations.
Can intersect with (ab^n)^{k+1} to get a language not weakly k-iterative.

Chinese Number Names

a. wu zhao zhao wu zhao
five trillion trillion five trillion

b. wu zhao zhao zhao zhao zhao wu zhao zhao zhao zhao wu zhao zhao zhao wu zhao zhao wu zhao

c. * wu zhao zhao wu zhao zhao zhao

d. wu zhao zhao zhao zhao wu zhao zhao

e. * wu zhao zhao wu zhao zhao wu zhao zhao zhao zhao

wu = 5

zhao = 10^{12}

$$J = \{\, wu\ zhao^{k_1}\ wu\ zhao^{k_2} \ldots wu\ zhao^{k_n} \mid k_1 > k_2 > \cdots > k_n > 0 \,\}.$$

¬ k-iterative for any k

Radzinski 1991

$J \in$ MCFL ?

$$J = \{\, wu\ zhao^{k_1}\ wu\ zhao^{k_2} \ldots wu\ zhao^{k_n} \mid k_1 > k_2 > \cdots > k_n > 0 \,\}.$$

$$\Psi(J) = \{\ (m, n) \mid m \geq 1, n \geq m(m+1)/2\ \}$$

¬ semilinear

Rank Hierarchy

rank (branching factor)

$$A(\alpha_1,\ldots,\alpha_m) \leftarrow B(x_1,\ldots,x_p),\ldots,D(z_1,\ldots,z_r)$$

- Each variable occurs at most once in $\alpha_1\ldots\alpha_m$
- nonterminal X = dim(X)-ary predicate on strings
- dim(S) = 1

m-MCFL(r)

dimension rank

Rank Hierarchy

2-MCFL(1) $\subsetneq$ 2-MCFL(2) = 2-MCFL(3) $\subsetneq$ 2-MCFL(4) $\subsetneq$ …

For $m \geq 3$,

m-MCFL(1) $\subsetneq$ m-MCFL(2) $\subsetneq$ m-MCFL(3) $\subsetneq$ m-MCFL(4) $\subsetneq$ …

Complexity of Recognition

	fixed language recognition	universal recognition
CFG	LOGCFL-complete	P-complete
TAG	LOGCFL-complete	P-complete
m-MCFG	LOGCFL-complete	NP-complete ($m \geq 2$)
MCFG	LOGCFL-complete	PSACE-complete/ EXPTIME-complete

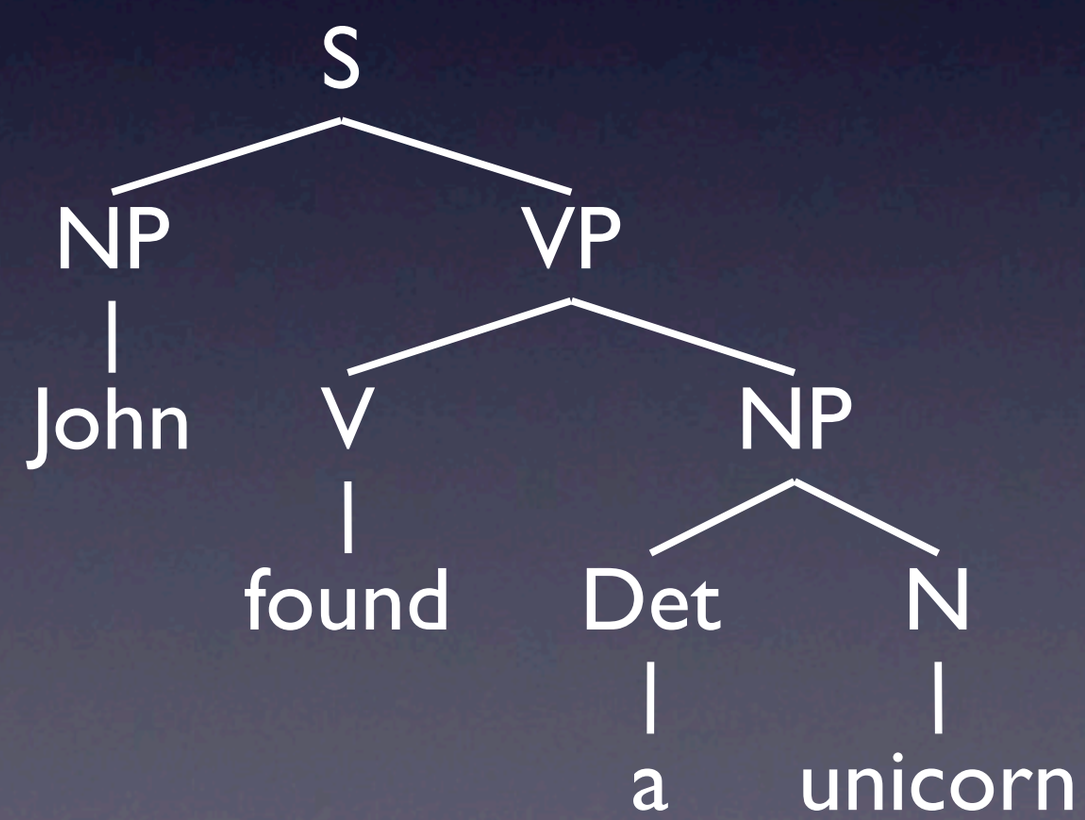
Satta 1992, Kaji, Nakanishi, Seki, and Kasami 1992

Fixed language recognition complexity is in LOGCFL all the way.

CFG recognition/parsing

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

John found a unicorn ∈ $L(G)$?



Well-known case of CFGs
Can associate an alternating algorithm with the grammar.

Alternation

$$A \rightarrow B\ C$$

$$A(xy) \leftarrow B(x), C(y)$$ Horn clause

Goal: $G \vdash A(v)$

Pick a rule $A(xy) \leftarrow B(x), C(y)$ $\exists$

Split v into v_1, v_2 $\exists$

Prove subgoals $B(v_1), C(v_2)$ $\forall$

When the goal is to prove $\vdash$ A(v), pick a rule, split v, and then prove subgoals.
Represent substrings by pairs of positions.

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

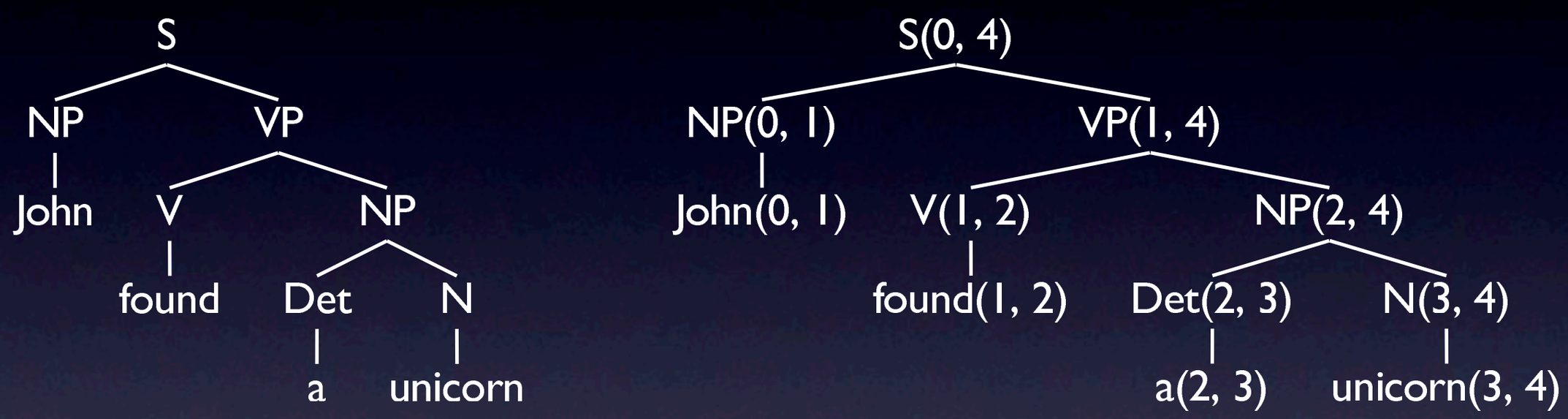
Definite clause grammar representation
Executable as Prolog code
:- is an alternative notation for ←

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

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

?- S(0, 4).

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 $CFTG_{IO}$ ⋮
Generation	CFG + Montague semantics†

† with a certain restriction

Parsing and generation as Datalog query evaluation

- Algorithms
  - Seminaive bottom-up ≈ CYK
  - Magic-sets rewriting ≈ Earley
- Computational complexity
  - Fixed grammar recognition
  - Uniform recognition
  - Parsing

Polynomial-time algorithm

```
Seminaive(P, D)
    agenda[0] ← D
    i ← 0
    chart ← ∅
    while agenda[i] ≠ ∅
        do
            chart ← chart ∪ agenda[i]
            agenda[i + 1] ← Conseq(P, agenda[i], chart) − chart
            i ← i + 1
    return chart
```

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

Computational complexity

- CFG

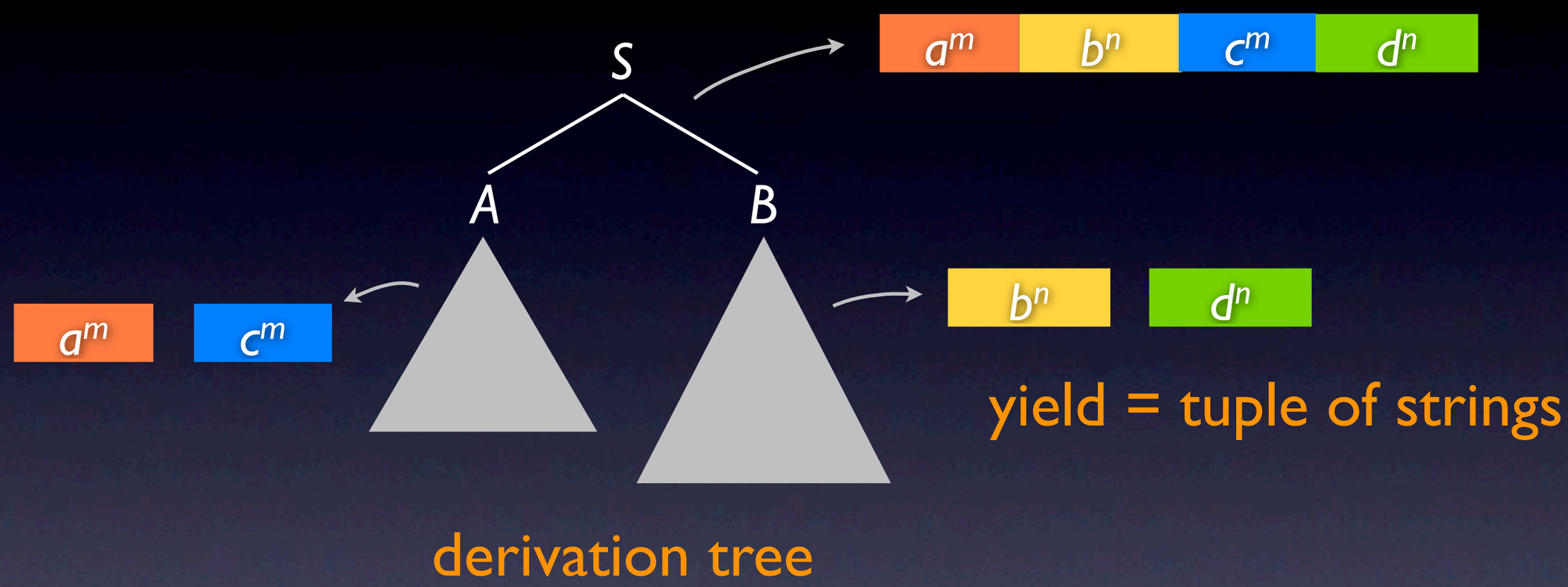
Fixed grammar recognition	LOGCFL-complete
Uniform recognition	P-complete
ε-free uniform recognition	LOGCFL-complete

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

$$AC^0 \subseteq NC^1 \subseteq L \subseteq NL \subseteq \text{LOGCFL} \subseteq AC^1 \subseteq NC^2 \subseteq P \subseteq NP$$

Multiple context-free grammars



$$S(x_1y_1x_2y_2) :- A(x_1,x_2), B(y_1,y_2).$$

$$A(\varepsilon,\varepsilon). \qquad B(\varepsilon,\varepsilon).$$

$$A(ax_1,cx_2) :- A(x_1,x_2). \qquad B(by_1,dy_2) :- B(y_1,y_2).$$

This is an example of a 2-MCFG.
An m-MCFG allows nonterminals to take up to m arguments.

$$S(x_1 y_1 x_2 y_2) :- A(x_1, x_2), B(y_1, y_2).$$

$$S(i, m) :- A(i, j, k, l), B(j, k, l, m).$$

$$A(ax_1, cx_2) :- A(x_1, x_2).$$

$$A(i, k, l, n) :- A(j, k, m, n), a(i, j), a(l, m).$$

Computational complexity

- MCFGs with fixed dimension and rank

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

dimension = arity of Datalog predicates
rank = number of subgoals in rules
ε-rule = Datalog rule with empty right-hand side

LOGCFL

- Characterized in terms of alternating Turing machines operating in log space with polynomial-size accepting computation trees

Ruzzo 1980

Alternating algorithm for Datalog query evaluation

- Each configuration is an attempt to prove some ground fact

∃ • Pick a matching rule

∃ • Pick constants that instantiate the rule

∀ • Prove all subgoals of the instantiated rule

- Can operate in log space if arity and number of subgoals are bounded

Datalog query evaluation

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

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

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

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

$NP(i_1, i_1, i_2) \coloneqq \textbf{John}(i_2).$

$V(i_1, i_3, i_2) \coloneqq \textbf{find}(i_1, i_3, i_2).$

$V(i_1, i_3, i_2) \coloneqq \textbf{catch}(i_1, i_3, i_2).$

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

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

$N(i_1, i_2) \coloneqq \textbf{unicorn}(i_1, i_2).$

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

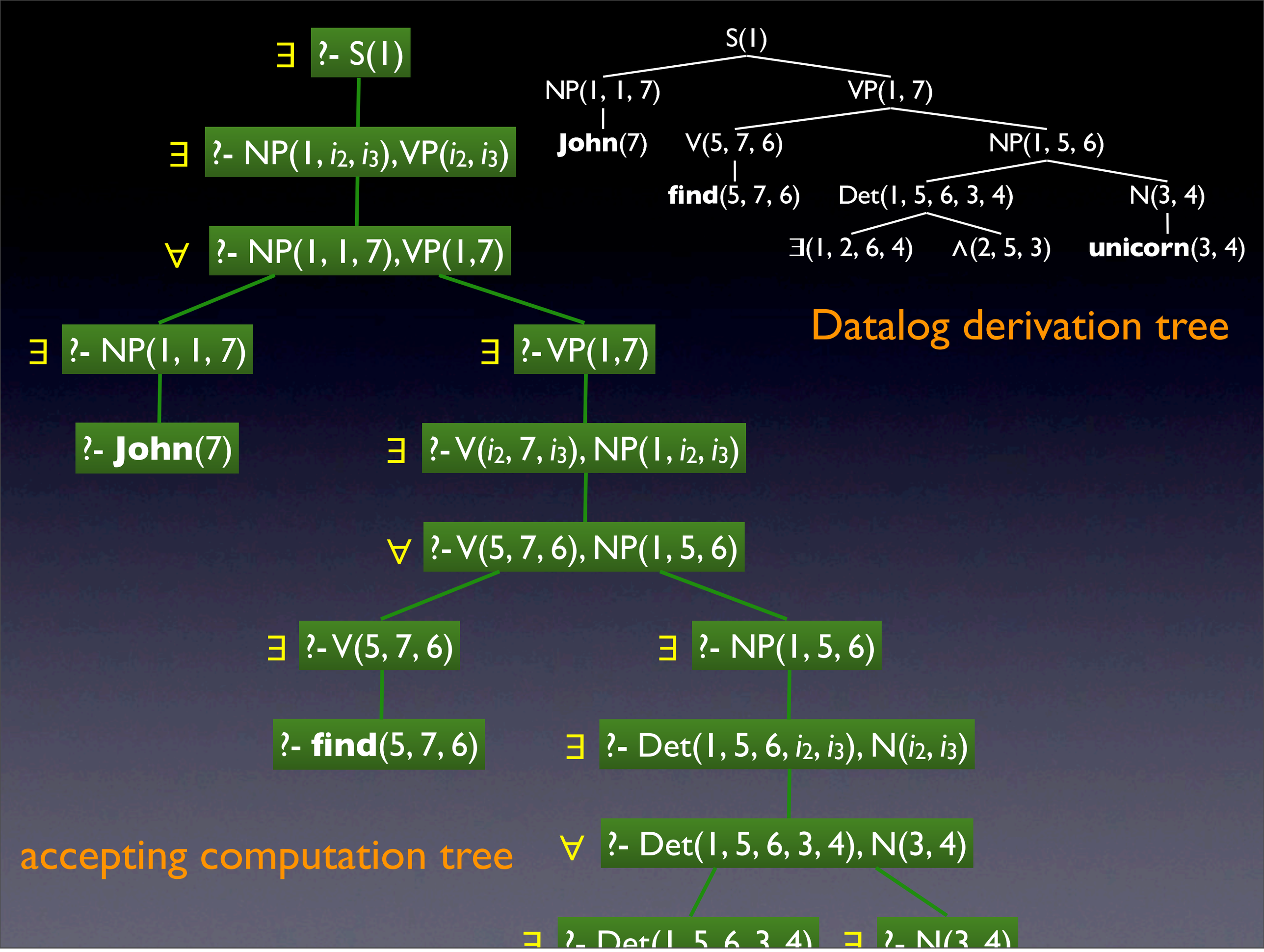
$\wedge(2, 5, 3).$

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

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

$\textbf{John}(6).$

$?\!- S(1).$



Datalog derivation tree

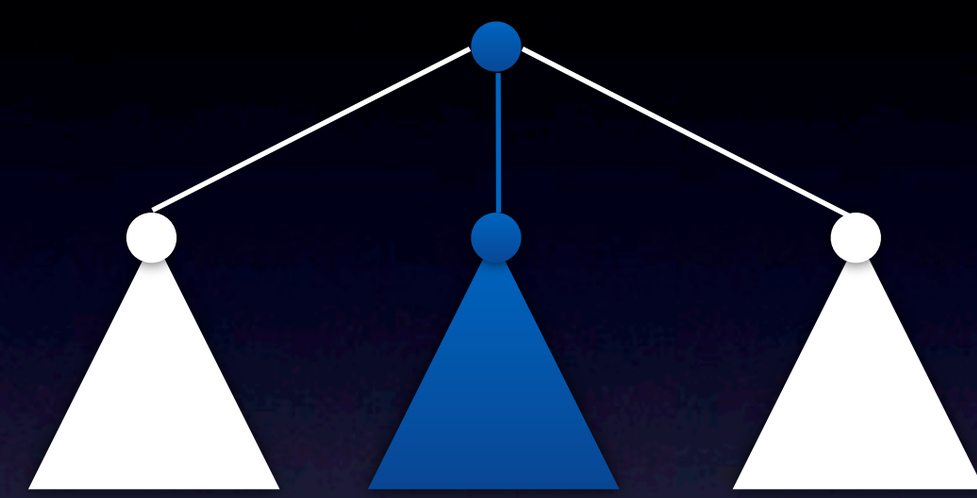
accepting computation tree

Size of Datalog derivation tree

Type 0

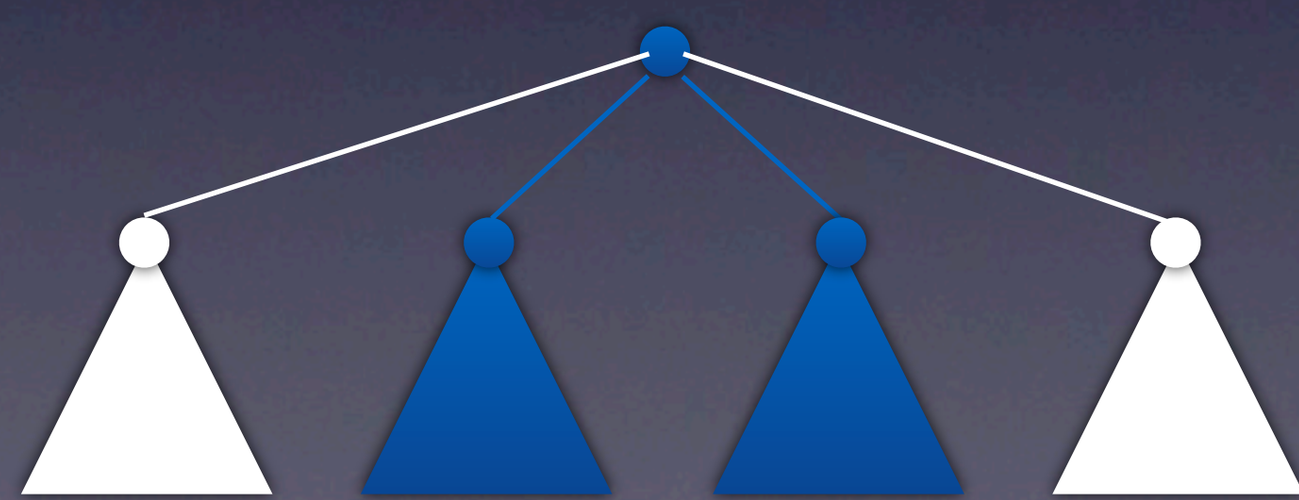
no database facts below

Type 1



all children except one are Type 0

Type 2



at least two children that are not Type 0

Number of nodes

n = number of extensional nodes
(nodes labeled by database facts)

- Type 0: exponential in the number of predicates, polynomial in n
- Type 1: linear in the number of predicates, polynomial in n
- Type 2: at most $n-1$

Alternating algorithm for MCFG recognition

G *w* string

log space

program(*G*) (database(*w*), query(*w*))

query(*w*)

derivation tree
from program(*G*) ∪ query(*w*)

The number of extensional nodes = length of *w*

Complexity of parsing

- MCFGs with fixed dimension and rank
- Output: parse forest = set of all ground rule instances useful for deriving the input query

Fixed grammar	FL^{LOGCFL}
Uniform	FP
ε-free uniform	FL^{LOGCFL}

Early parsing

- From CFGs to MCFGs

Polynomial-time algorithm

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

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

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

Seminaive is a purely bottom-up tabular algorithm.
This algorithm works for Datalog programs in general.
The size of chart is $O(n^r)$ (r = maximal arity)
Running time $O(n^k)$ depends on the number of variables in rules.

S → NP VP
S → Aux NP VP
S → VP
NP → Det N1
NP → Name
NP → Pronoun
VP → V
VP → V NP
VP → V NP PP
VP → VP PP
N1 → N
N1 → Name N1
N1 → N1 PP
PP → P NP

Aux → does
Det → that | this | a | the
Name → Houston | TWA
Pronoun → I | she | me
V → book | include | prefer
N → book | flight | meal | money
P → from | to | on

An example of a context-free grammar.
Adapted from Jurafsky and Martin.

S(*i*, *k*) :- NP(*i*, *j*),VP(*j*, *k*).
S(*i*, *l*) :- Aux(*i*, *j*), NP(*j*, *k*),VP(*k*, *l*).
S(*i*, *j*) :- VP(*i*, *j*).
NP(*i*, *k*) :- Det(*i*, *j*), N1(*j*, *k*).
NP(*i*, *j*) :- Name(*i*, *j*).
NP(*i*, *j*) :- Pronoun(*i*, *j*).
VP(*i*, *j*) :- V(*i*, *j*).
VP(*i*, *k*) :- V(*i*, *j*), NP(*j*, *k*).
VP(*i*, *l*) :- V(*i*, *j*), NP(*j*, *k*), PP(*k*, *l*).
VP(*i*, *k*) :- VP(*i*, *j*), PP(*j*, *k*).
N1(*i*, *j*) :- N(*i*, *j*).
N1(*i*, *k*) :- N1(*i*, *j*), N(*j*, *k*).
N1(*i*, *k*) :- N1(*i*, *j*), PP(*j*, *k*).
PP(*i*, *k*) :- P(*i*, *j*), NP(*j*, *k*).

Aux(*i*, *j*) :- does(*i*, *j*).
Det(*i*, *j*) :- that(*i*, *j*).
Det(*i*, *j*) :- this(*i*, *j*).
Det(*i*, *j*) :- a(*i*, *j*).
Det(*i*, *j*) :- the(*i*, *j*).
Name(*i*, *j*) :- Houston(*i*, *j*).
Name(*i*, *j*) :- TWA(*i*, *j*).
Pronoun(*i*, *j*) :- I(*i*, *j*).
Pronoun(*i*, *j*) :- she(*i*, *j*).
Pronoun(*i*, *j*) :- me(*i*, *j*).
V(*i*, *j*) :- book(*i*, *j*).
V(*i*, *j*) :- include(*i*, *j*).
V(*i*, *j*) :- prefer(*i*, *j*).
N(*i*, *j*) :- book(*i*, *j*).
N(*i*, *j*) :- flight(*i*, *j*).
N(*i*, *j*) :- meal(*i*, *j*).
N(*i*, *j*) :- money(*i*, *j*).
P(*i*, *j*) :- from(*i*, *j*).
P(*i*, *j*) :- to(*i*, *j*).
P(*i*, *j*) :- on(*i*, *j*).

Datalog program representing CFG.
Running time of Seminaive is $O(n^4)$

D = { book(0, 1), the(1, 2), flight(2, 3), from(3, 4), Houston(4, 5) }.

agenda[0] = { book(0, 1), the(1, 2), flight(2, 3), from(3, 4), Houston(4, 5) }.
agenda[1] = { N(0, 1),V(0, 1), Det(1, 2), N(2, 3), P(3, 4), Name(4, 5) }
agenda[2] = { N1(0, 1),VP(0, 1), N1(2, 3), NP(4, 5) }
agenda[3] = { NP(1, 3), PP(3, 5) }
agenda[4] = {VP(0, 3),VP(0, 5), N1(2, 5) }
agenda[5] = { S(0, 3), S(0, 5), NP(1, 5) }
agenda[6] = ∅

$$chart = agenda[0] \cup \cdots \cup agenda[6]$$

The run of Seminaive on “book the flight from Houston”

Outputting shared forest

Seminaive_parse(**P**, D)

1 $agenda[0] \leftarrow D$
2 $i \leftarrow 0$
3 $chart \leftarrow \varnothing$
4 $parse \leftarrow \varnothing$
5 **while** $agenda[i] \neq \varnothing$
6 **do**
7 $chart \leftarrow chart \cup agenda[i]$
8 $agenda[i+1] \leftarrow \text{Conseq}(\mathbf{P}, agenda[i], chart) - chart$
9 $parse \leftarrow parse \cup \text{Inst}(\mathbf{P}, agenda[i], chart)$
10 $i \leftarrow i + 1$
11 **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*

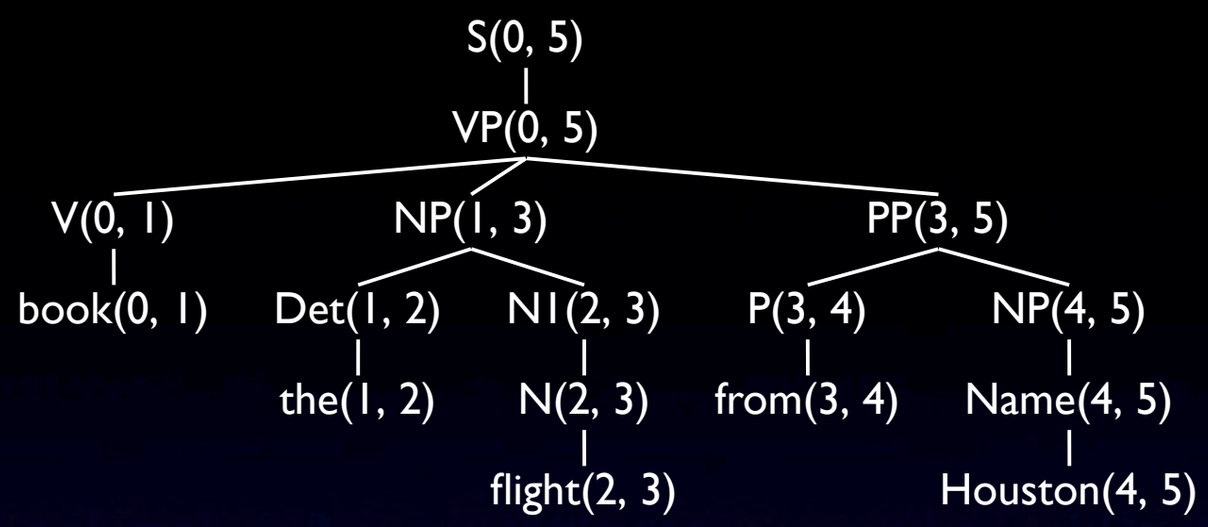
instances of rules with right-hand side consisting of one fact in *agenda*[i] and some facts in *chart*

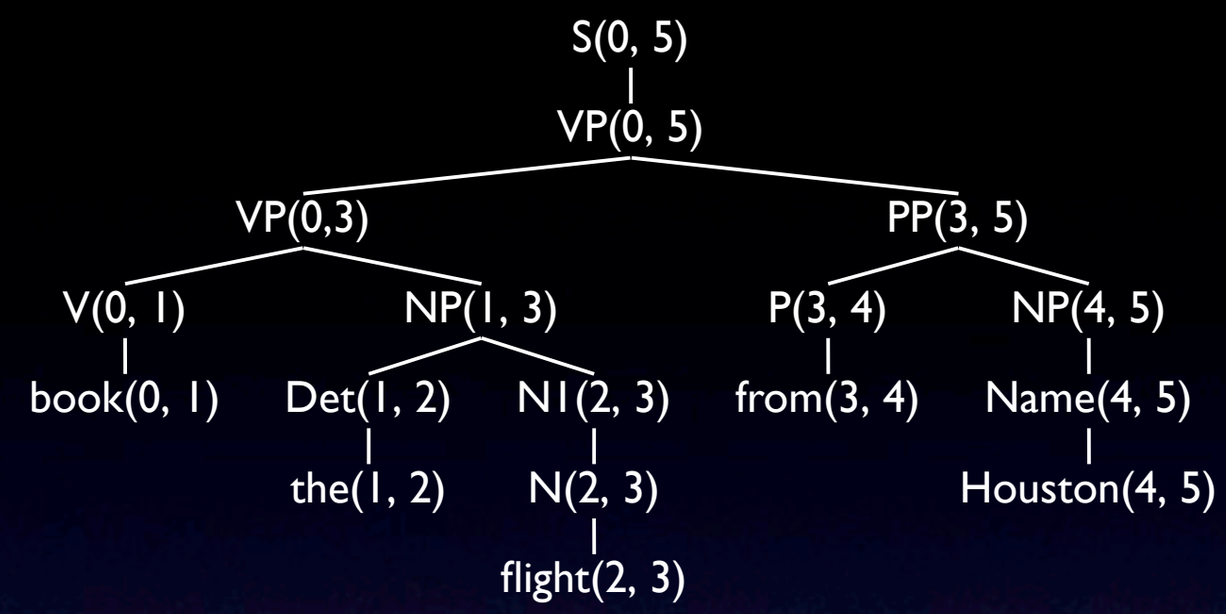
shared parse forest

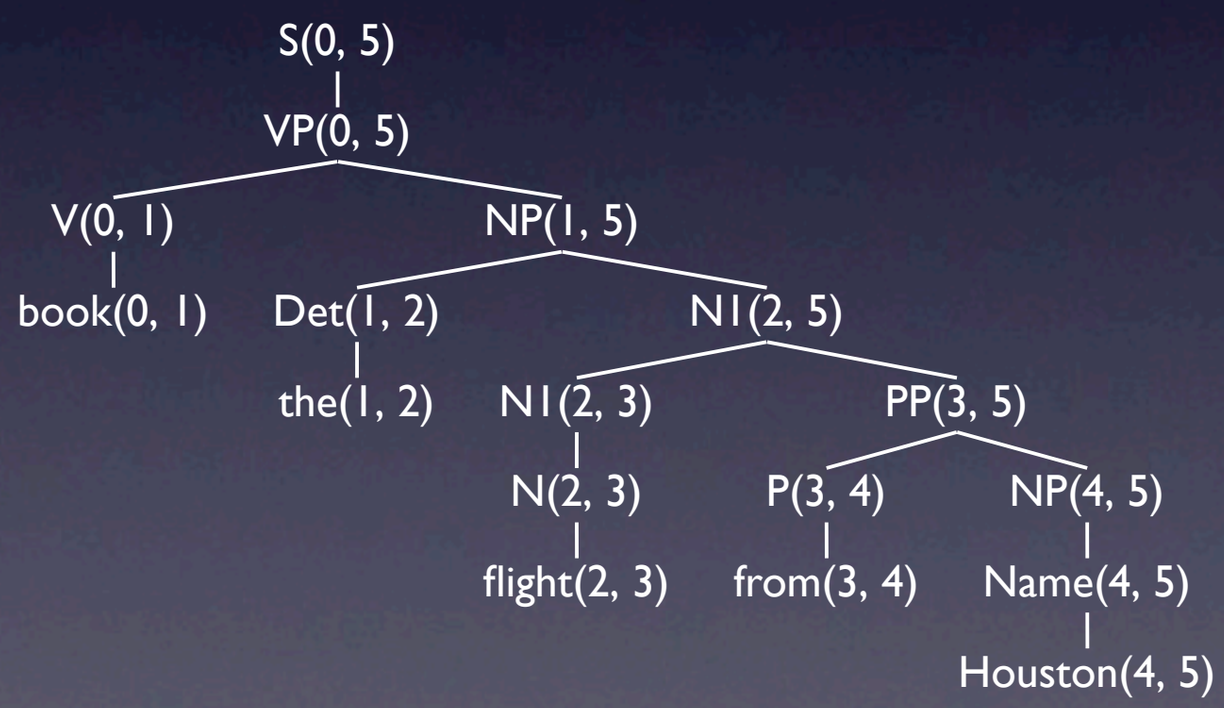
Parsing algorithms are often not explicitly stated in textbooks.
Lines 8 and 9 should be performed simultaneously.

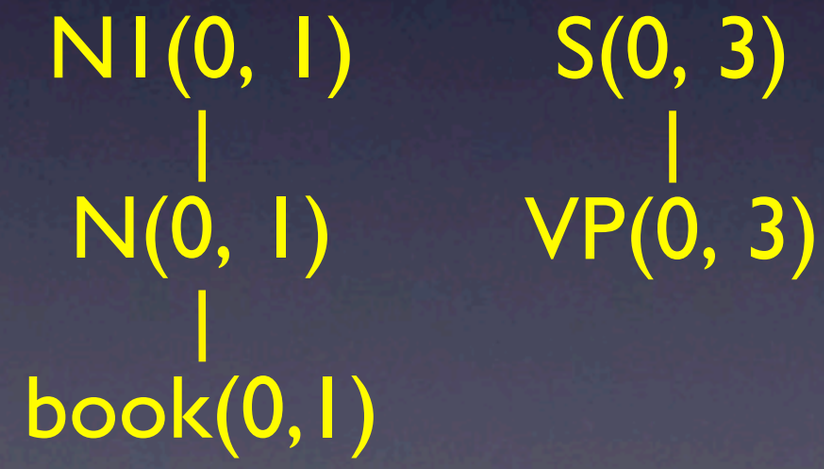
```
parse = { book(0, 1).  the(1, 2).  flight(2, 3).  from(3, 4).  Houston(4, 5).
          N(0, 1) :- book(0, 1).
          V(0, 1) :- book(0, 1).
          Det(1, 2) :- the(1, 2).
          N(2, 3) :- flight(2, 3).
          P(3, 4) :- from(3, 4).
          Name(4, 5) :- Houston(4, 5).
          N1(0, 1) :- N(0, 1).
          VP(0, 1) :- V(0, 1).
          N1(2, 3) :- N(2, 3).
          NP(4, 5) :- Name(4, 5).
          NP(1, 3) :- Det(1, 2), N1(2, 3).
          PP(3, 5) :- P(3, 4), NP(4, 5).
          VP(0, 3) :- V(0, 1), NP(1, 3).
          VP(0, 5) :- V(0, 1), NP(1, 3), PP(3, 5).
          N1(2, 5) :- N1(2, 3), PP(3, 5).
          S(0, 3) :- VP(0, 3).
          S(0, 5) :-VP(0, 5).
          VP(0, 5) :-VP(0, 3), PP(3, 5).
          NP(1, 5) :- Det(1, 2), N1(2, 5).
          VP(0, 5) :- V(0, 1), NP(1, 5).
        }
```


Output parse forest.
Can be thought of as a CFG for a singleton language.





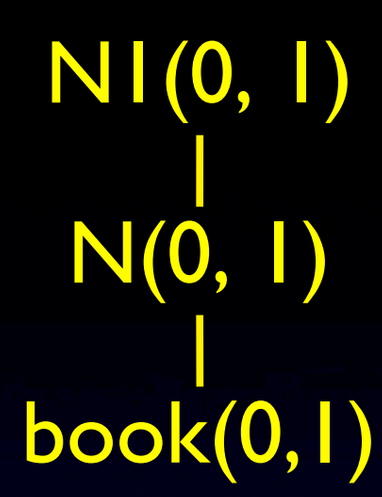


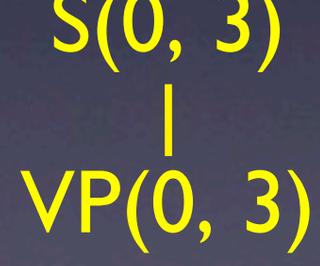


Some rule instances found by the algorithm cannot be used in any derivation trees.

parse = { book(0, 1). the(1, 2). flight(2, 3). from(3, 4). Houston(4, 5).

N(0, 1) :- book(0, 1).
V(0, 1) :- book(0, 1).
Det(1, 2) :- the(1, 2).
N(2, 3) :- flight(2, 3).
P(3, 4) :- from(3, 4).
Name(4, 5) :- Houston(4, 5).
N1(0, 1) :- N(0, 1).
VP(0, 1) :- V(0, 1).
N1(2, 3) :- N(2, 3).
NP(4, 5) :- Name(4, 5).
NP(1, 3) :- Det(1, 2), N1(2, 3).
PP(3, 5) :- P(3, 4), NP(4, 5).
VP(0, 3) :- V(0, 1), NP(1, 3).
VP(0, 5) :- V(0, 1), NP(1, 3), PP(3, 5).
N1(2, 5) :- N1(2, 3), PP(3, 5).
S(0, 3) :- VP(0, 3).
S(0, 5) :-VP(0, 5).
VP(0, 5) :-VP(0, 3), PP(3, 5).
NP(1, 5) :- Det(1, 2), N1(2, 5).
VP(0, 5) :- V(0, 1), NP(1, 5).
}





This parse forest is not “reduced”.

D = { book(0, 1), the(1, 2), flight(2, 3), from(3, 4), Houston(4, 5) }.

agenda[0] = { book(0, 1), the(1, 2), flight(2, 3), from(3, 4), Houston(4, 5) }.
agenda[1] = { N(0, 1), V(0, 1), Det(1, 2), N(2, 3), P(3, 4), Name(4, 5) }
agenda[2] = { N1(0, 1), VP(0, 1), N1(2, 3), NP(4, 5) }
agenda[3] = { NP(1, 3), PP(3, 5) }
agenda[4] = { VP(0, 3), VP(0, 5), N1(2, 5) }
agenda[5] = { S(0, 3), S(0, 5), VP(0, 5), NP(1, 5) }
agenda[6] = ∅

$$chart = agenda[0] \cup \cdots \cup agenda[6]$$

"Weak parse" as opposed to "parse" (Ruzzo).

Earley parsing

- $O(n^3)$ running time for arbitrary CFG
- Partial reduction of parse forest by top-down filtering
- Correct prefix property

The first two goals independent of each other in principle.
Top-down filtering plus appropriate control achieves partial reduction.

Correct prefix property

- Process input from left to right
- Reject the input as soon as the portion of the input that has been processed so far is not a correct prefix (i.e., cannot be extended to an element of the language)

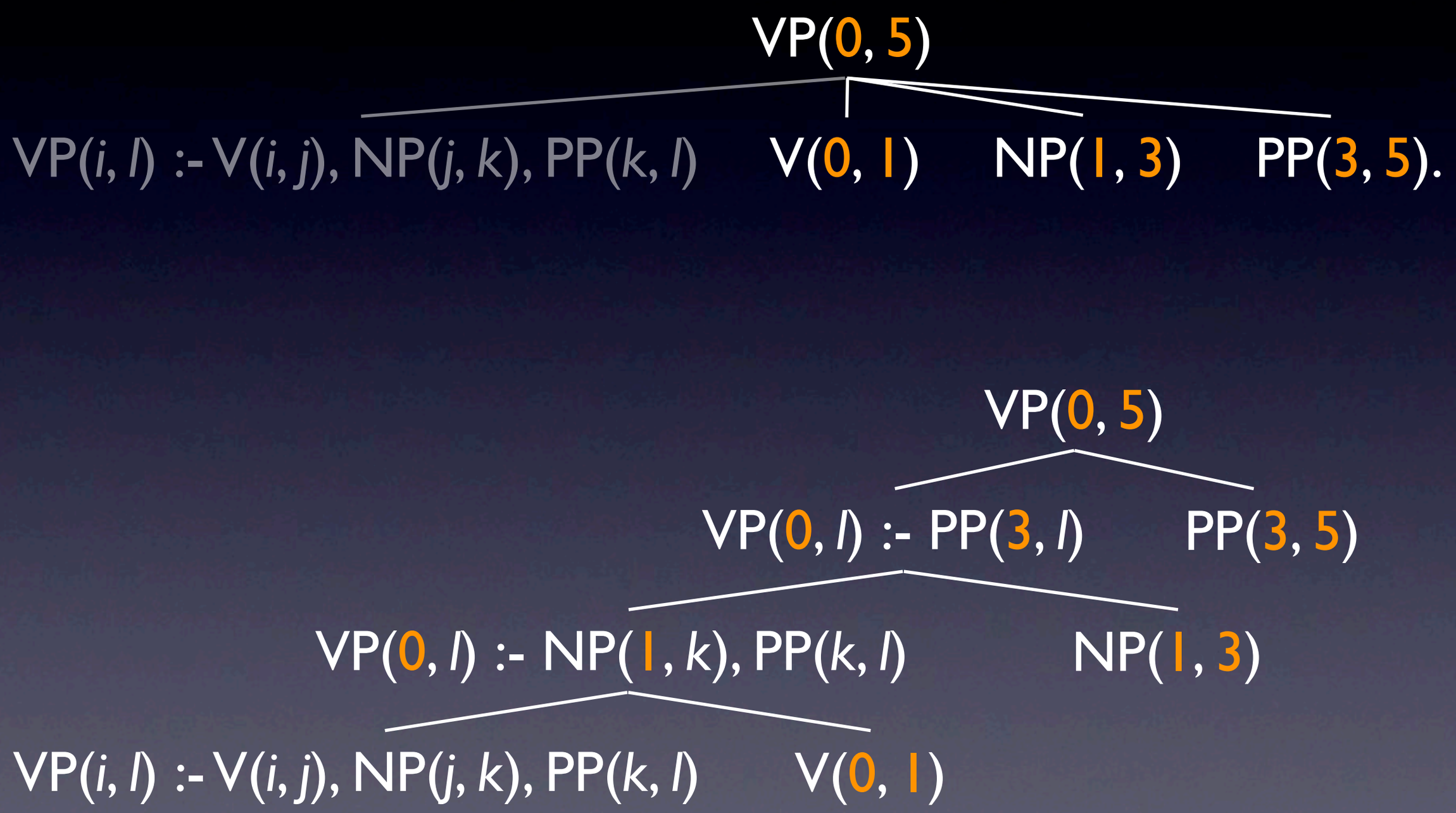
Term comes from stack-based parsing.

Earley parsing

- Binarization of rules
- Additional subgoals

Binarization improves running time.
Additional subgoals achieve top-down filtering.
Binarization helps for top-down filtering as well.
The way the top-down filtering is done achieves prefix-correctness.

Binarization of rules



Derived facts and derived clauses.

Dotted rules

VP → V NP PP

VP → V • NP PP VP → V NP • PP

VP(0, 5)

VP(0, *l*) :- PP(3, *l*) PP(3, 5)

VP(0, *l*) :- NP(1, *k*), PP(*k*, *l*) NP(1, 3)

VP(*i*, *l*) :- V(*i*, *j*), NP(*j*, *k*), PP(*k*, *l*) V(0, 1)

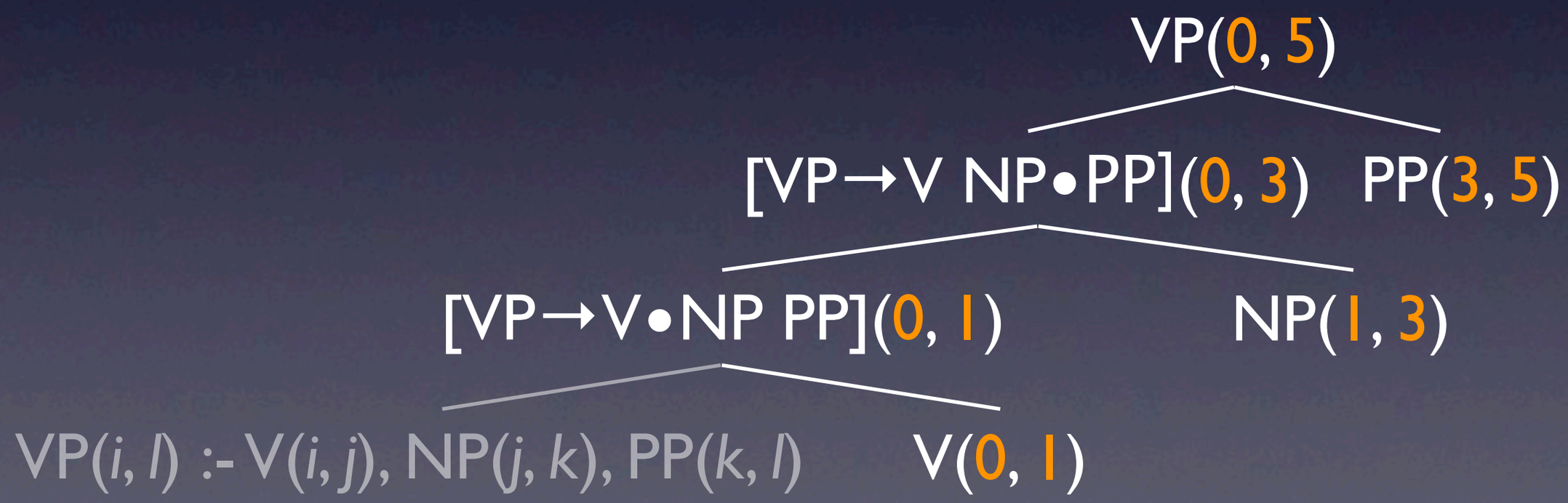
Derived clauses -> derived facts
Dotted rules as new predicate names.

Dotted rules

[VP→V•NP PP](i, j) :- V(i,j).

[VP→V NP•PP](i, k) :- [VP→V•NP PP](i, j), NP(j, k).

VP(i, l) :- [VP→V NP•PP](i, k), PP(k, l).

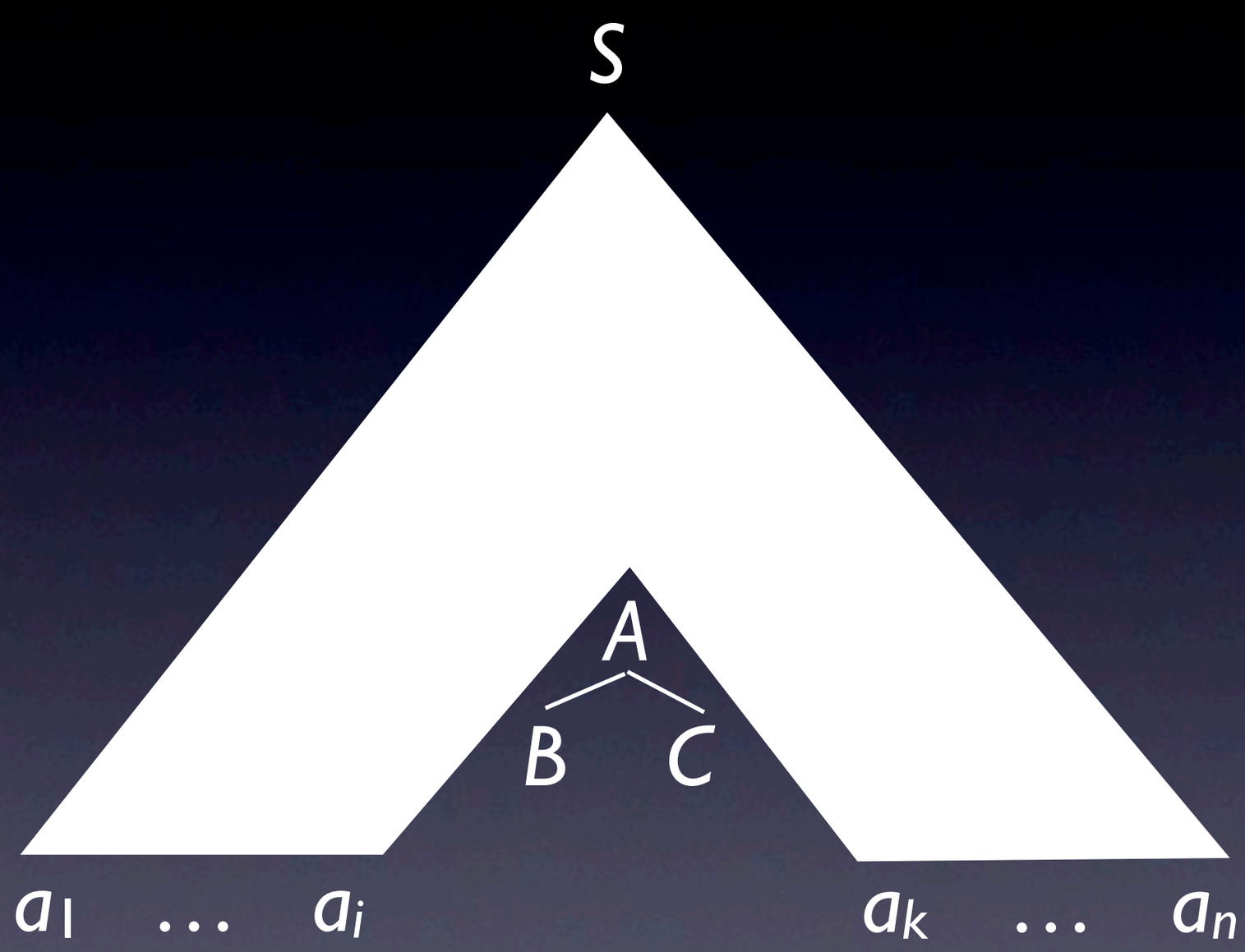


$[S \rightarrow NP \bullet VP](i, j)$:- $NP(i, j)$.
$S(i, k)$:- $[S \rightarrow NP \bullet VP](i, j), VP(j, k)$.
$[S \rightarrow Aux \bullet NP\ VP](i, j)$:- $Aux(i, j)$.
$[S \rightarrow Aux\ NP \bullet VP](i, k)$:- $[S \rightarrow Aux \bullet NP\ VP](i, j), Aux(j, k)$.
$S(i, l)$:- $[S \rightarrow Aux\ NP \bullet VP](i, k), VP(k, l)$.
$S(i, j)$:- $VP(i, j)$.
$[NP \rightarrow Det \bullet N1](i, j)$:- $Det(i, j)$.
$NP(i, k)$:- $[NP \rightarrow Det \bullet N1](i, j), N1(j, k)$.
$NP(i, j)$:- $Name(i, j)$.
$NP(i, j)$:- $Pronoun(i, j)$.
$VP(i, j)$:- $V(i, j)$.
$[VP \rightarrow V \bullet NP](i, j)$:- $V(i, j)$.
$VP(i, k)$:- $[VP \rightarrow V \bullet NP](i, j), NP(j, k)$.
$[VP \rightarrow V \bullet NP\ PP](i, j)$:- $V(i, j)$.
$[VP \rightarrow V\ NP \bullet PP](i, k)$:- $[VP \rightarrow V \bullet NP\ PP](i, j), NP(j, k)$.
$VP(i, l)$:- $[VP \rightarrow V\ NP \bullet PP](i, k), PP(k, l)$.

⋮

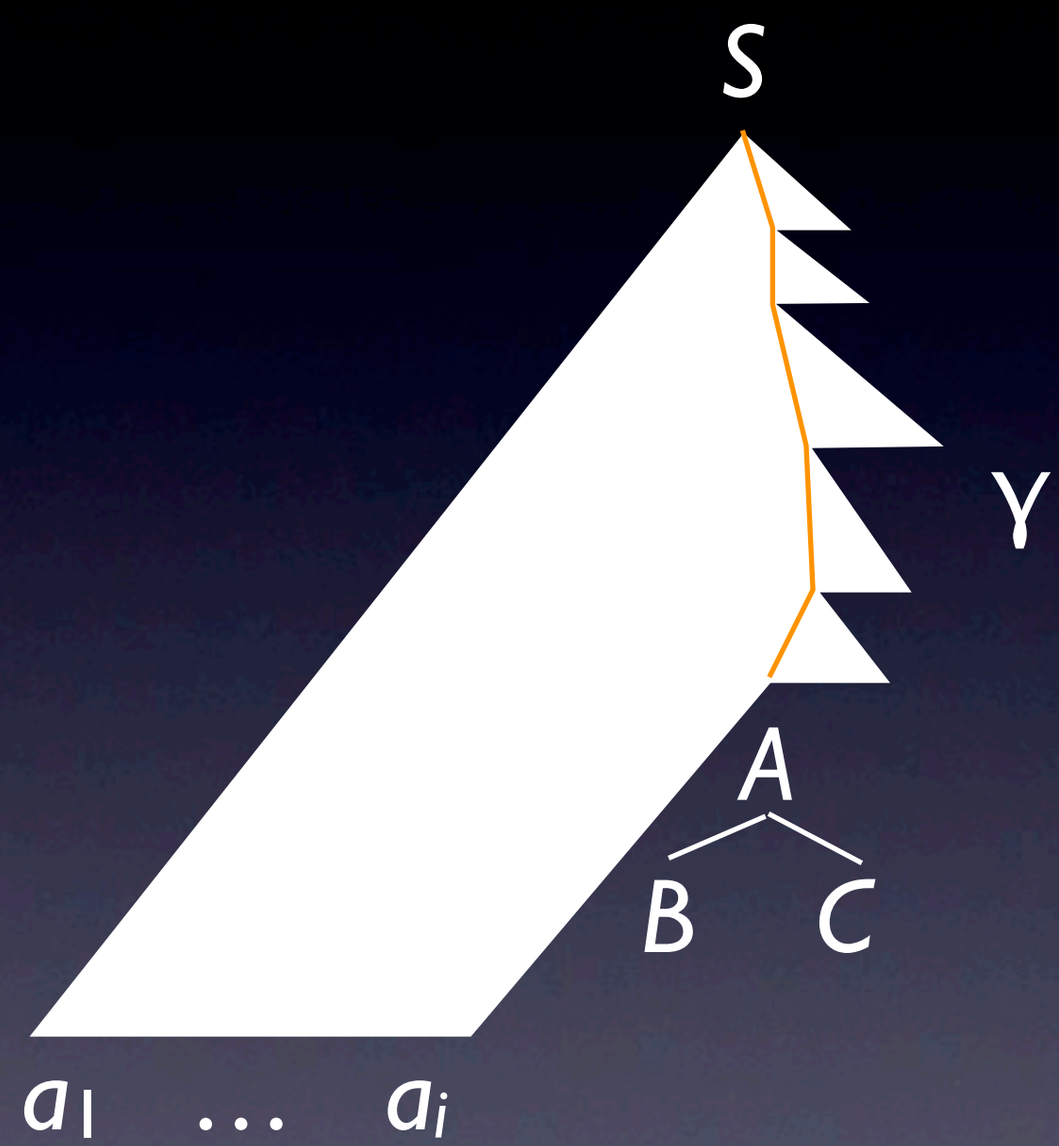
Binarized program: each rule contains at most 3 variables.
Running time is $O(n^3)$
"Bottom-up chart" recognizer

Top-down filtering



The existence of the white region needs to be checked in order to produce a complete parse/ reduced parse forest.
But this is too much and inconsistent with correct prefix property.

Top-down filtering



$S \Rightarrow^* a_1 \ldots a_i \, A \, \gamma$ leftmost derivation

Require there to be a path from the root and the part to the left of the path to be realized. This has the dual effect of partial reduction of parse forest and correct prefix property.

Top-down prediction

$$m\_VP(i) \Leftrightarrow S \Rightarrow^* a_1 \ldots a_i \text{ VP } \gamma$$

"VP is predicted to start at *i*"

[VP→V•NP PP](*i*, *j*) :- *m*_VP(*i*),V(*i*,*j*).

[VP→V NP•PP](*i*, *k*) :- [VP→V•NP PP](*i*, *j*), NP(*j*, *k*).

VP(*i*, *l*) :- [VP→V NP•PP](*i*, *k*), PP(*k*, *l*).

*m*_S(0). seed

S(*i*, *j*) :- VP(*i*, *j*). *m*_VP(*i*) :- m_S(*i*).

*m*_V(*i*) :- *m*_VP(*i*).

*m*_NP(*j*) :- [VP→V•NP PP](*i*, *j*).

*m*_PP(*k*) :- [VP→V NP•PP](*i*, *k*).

Earley deduction system

$m\_S(0).$ INITIALIZE

$m\_B(j)$:- $[A \rightarrow \alpha \bullet B\beta](i, j).$

$m\_B(i)$:- $m\_A(i).$ $(A \rightarrow B\beta)$ PREDICT

$[A \rightarrow B \bullet \beta](i, j)$:- $m\_A(i), B(i, j).$

$[A \rightarrow \alpha B \bullet \beta](i, k)$:- $[A \rightarrow \alpha \bullet B\beta](i, j), B(j, k).$

$A(i, k)$:- $[A \rightarrow \alpha \bullet B](i, j), B(j, k).$ COMPLETE

$[A \rightarrow b \bullet \beta](i, j)$:- $m\_A(i), b(i, j).$

$[A \rightarrow \alpha b \bullet \beta](i, k)$:- $[A \rightarrow \alpha \bullet b\beta](i, j), b(j, k).$

$A(i, k)$:- $[A \rightarrow \alpha \bullet b](i, j), b(j, k).$ SCAN

A minor variation, expressed in the style of Datalog.
“m_” comes from magic-sets rewriting.

$m\_NP(i)$:- $m\_S(i)$.
$[S \rightarrow NP \bullet VP](i, j)$:- $m\_S(i)$, $NP(i, j)$.
$m\_VP(j)$:- $[S \rightarrow NP \bullet VP](i, j)$.
$S(i, k)$:- $[S \rightarrow NP \bullet VP](i, j)$, $VP(j, k)$.
$m\_Aux(i)$:- $m\_S(i)$.
$[S \rightarrow Aux \bullet NP\ VP](i, j)$:- $m\_S(i)$, $Aux(i, j)$.
$m\_NP(j)$:- $[S \rightarrow Aux \bullet NP\ VP](i, j)$.
$[S \rightarrow Aux\ NP \bullet VP](i, k)$:- $[S \rightarrow Aux \bullet NP\ VP](i, j)$, $Aux(j, k)$.
$m\_VP(k)$:- $[S \rightarrow Aux\ NP \bullet VP](i, k)$.
$S(i, l)$:- $[S \rightarrow Aux\ NP \bullet VP](i, k)$, $VP(k, l)$.
$m\_VP(i)$:- $m\_S(i)$.
$S(i, j)$:- $m\_S(i)$, $VP(i, j)$.
$m\_Det(i)$:- $m\_NP(i)$.
$[NP \rightarrow Det \bullet N1](i, j)$:- $m\_NP(i)$, $Det(i, j)$.
$m\_N1(j)$:- $[NP \rightarrow Det \bullet N1](i, j)$.
$NP(i, k)$:- $[NP \rightarrow Det \bullet N1](i, j)$, $N1(j, k)$.

⋮

Earley deduction system as Datalog program.

Prefix-correct control algorithm

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

$agenda \leftarrow D$
$chart \leftarrow \varnothing$
while $agenda \neq \varnothing$
do $trigger \leftarrow \text{Pop}(agenda)$
$chart \leftarrow chart \cup \{trigger\}$
$new \leftarrow \text{Conseq}(\mathbf{P}, \{trigger\}, chart) - chart$

foreach $item \in new$
do $\text{Push}(agenda, item)$
return $chart$

Agenda is now first-in last-out.
Push and pop one item at a time.

Earley = magic-sets rewriting

$m\_S(0).$

$m\_B(j)$:- $sup_{r,n}(i, j)$. $(r = A \rightarrow \alpha B \beta, n = |\alpha|)$

$m\_B(i)$:- $m\_A(i)$. $(A \rightarrow B\beta)$

$sup_{r,1}(i, j)$:- $m\_A(i), B(i, j)$. $(r = A \rightarrow B\beta)$

$sup_{r,n+1}(i, k)$:- $sup_{r,n}(i, j), B(j, k)$. $(r = A \rightarrow \alpha B \beta, n = |\alpha|)$

$A(i, k)$:- $sup_{r,n}(i, j), B(j, k)$. $(r = A \rightarrow \alpha \bullet B, n = |\alpha|)$

$sup_{r,1}(i, j)$:- $m\_A(i), b(i, j)$. $(r = A \rightarrow b\beta)$

$sup_{r,n+1}(i, k)$:- $sup_{r,n}(i, j), b(j, k)$. $(r = A \rightarrow \alpha b \beta, n = |\alpha|)$

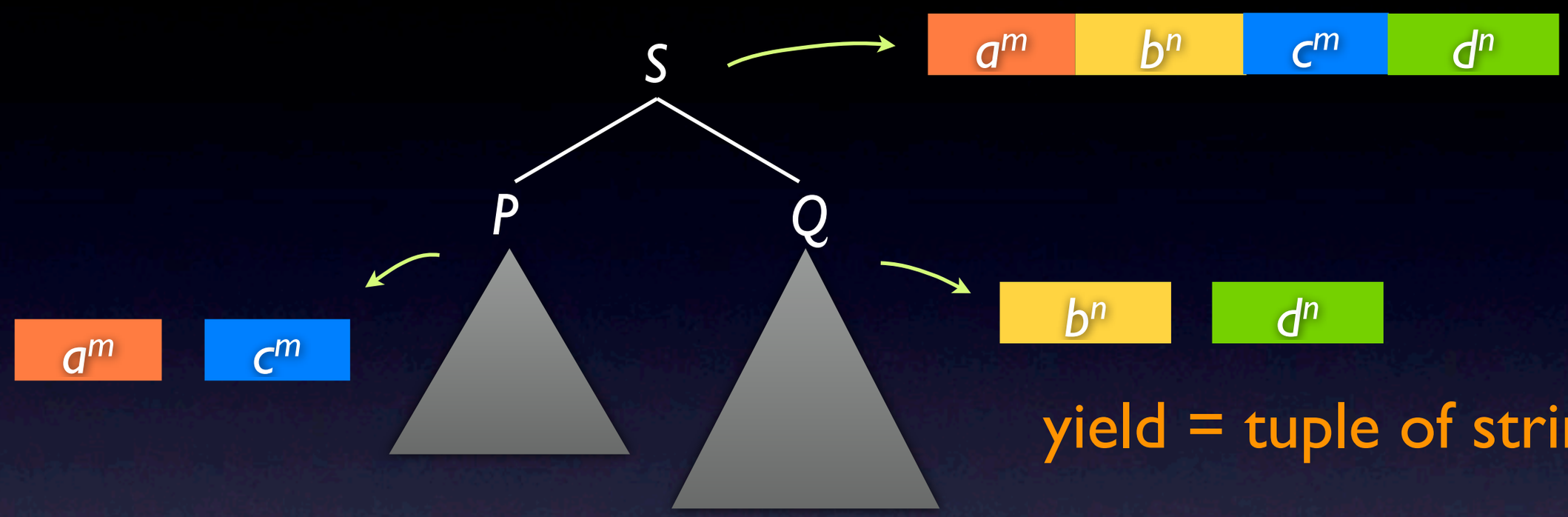
$A(i, k)$:- $sup_{r,n}(i, j), b(j, k)$. $(r = A \rightarrow \alpha \bullet b, n = |\alpha|)$

Notation for magic-sets rewriting.
Supplementary predicates correspond to dotted rules.

Magic-sets rewriting

- Logic program transformation particularly effective for Datalog
- Rougly equivalent to "memoized" top-down evaluation methods (including Earley deduction)
- Binarization of rules, top-down filtering

Multiple context-free grammars



yield = tuple of strings

derivation tree

$$S(x_1 y_1 x_2 y_2) :- P(x_1, x_2), Q(y_1, y_2).$$

bottom-up

$$P(a,c). \qquad Q(b,d).$$

$$P(ax_1, cx_2) :- P(x_1, x_2). \qquad Q(by_1, dy_2) :- Q(y_1, y_2).$$

A slight variation of a by-now-familiar 2-MCFG.
Generates { $a^m b^n c^m d^n$ | m,n ≥1 }
Convert to Datalog.

$$S(x_1 y_1 x_2 y_2) :- P(x_1, x_2), Q(y_1, y_2).$$

$$S(i, m) :- P(i, j, k, l), Q(j, k, l, m).$$

$$P(ax_1, cx_2) :- P(x_1, x_2).$$

$$P(i, k, l, n) :- P(j, k, m, n), a(i, j), a(l, m).$$

From MCFG to Datalog

$S(\ x_1\ y_1\ x_2\ y_2\ ) :- P(x_1, x_2), Q(y_1, y_2).$
$\quad i\ \ j\ \ k\ \ l\ \ m$

$P(\ a\ ,\ c\ ).$
$\quad i\ j\ k\ l$

$P(\ a\, x_1\ ,\ c\, x_2\ ) :- P(x_1, x_2).$
$\quad i\ j\ \ k\ lm\ \ n$

$Q(b, d).$

$Q(by_1, dy_2) :- Q(y_1, y_2).$

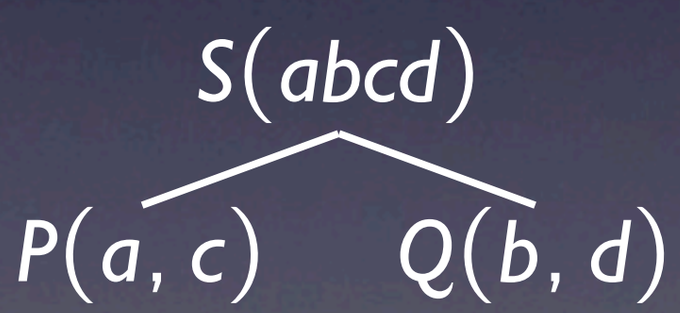
$S(i, m) :- P(i, j, k, l), Q(j, k, l, m).$

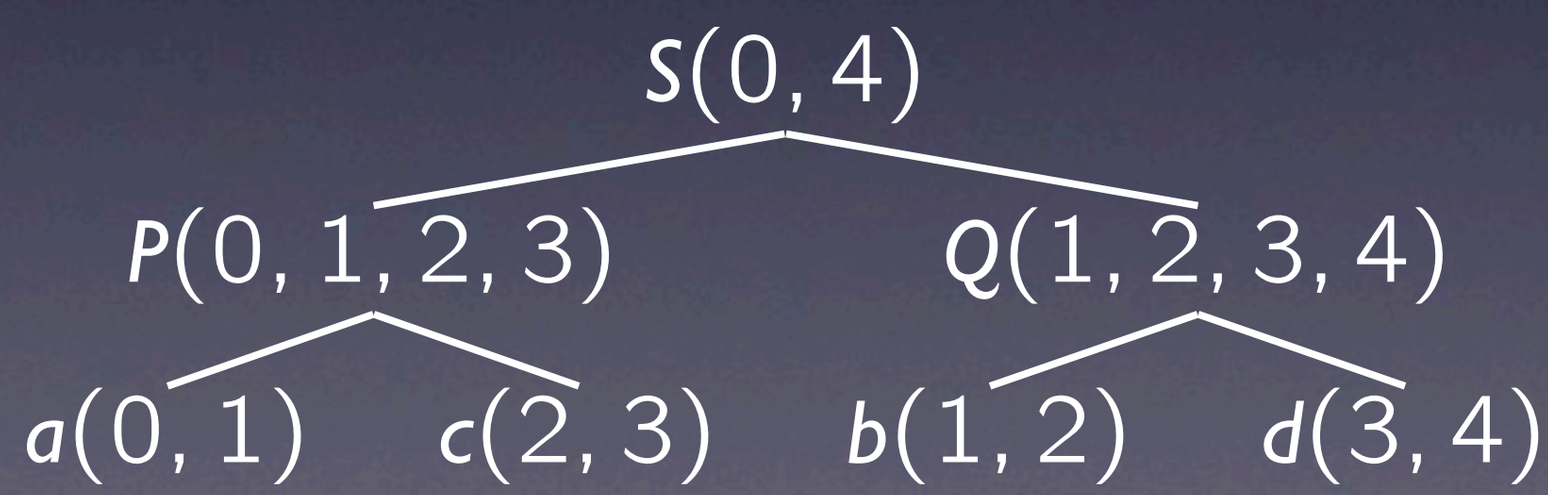
$P(i, j, k, l) :- a(i, j), c(k, l).$

$P(i, k, l, n) :- a(i, j), P(j, k, m, n), c(l, m).$

$Q(i, j, k, l) :- b(i, j), d(k, l).$

$Q(i, k, l, n) :- b(i, j), d(l, m), Q(j, k, m, n).$





Adornment

$$1\colon S^{bf}(i,m) :- P^{bfff}(i,j,k,l), Q^{bbbf}(j,k,l,m).$$
$$2\colon P^{bfff}(i,j,k,l) :- a^{bf}(i,j), c^{ff}(k,l).$$
$$3\colon P^{bfff}(i,k,l,n) :- a^{bf}(i,j), P^{bfff}(j,k,m,n), c^{fb}(l,m).$$
$$4\colon Q^{bbbf}(i,j,k,l) :- b^{bb}(i,j), d^{bf}(k,l).$$
$$5\colon Q^{bbbf}(i,k,l,n) :- b^{bf}(i,j), d^{bf}(l,m), Q^{bbbf}(j,k,m,n).$$

SLD derivation

$$?- S(0,x).$$
$$\stackrel{1}{\Longrightarrow} ?- P(0,j_1,k_1,l_1), Q(j_1,k_1,l_1,x).$$
$$\stackrel{2}{\Longrightarrow} ?- a(0,j_1), c(k_1,l_1), Q(j_1,k_1,l_1,x).$$
$$\stackrel{a(0,1)}{\Longrightarrow} ?- c(k_1,l_1), Q(1,k_1,l_1,x).$$
$$\stackrel{c(2,3)}{\Longrightarrow} ?- Q(1,2,3,x).$$
$$\stackrel{4}{\Longrightarrow} ?- b(1,2), d(3,x).$$
$$\stackrel{b(1,2)}{\Longrightarrow} ?- d(3,x).$$
$$\stackrel{d(3,4)}{\Longrightarrow} ?- [].$$

Applying magic-sets rewriting.
Each argument position is marked as “free” (uninstantiated) or “bound” (instantiated), according to the status of arguments in SLD derivations.
Initial query is assumed to be S(0,x).
SLD derivations generalize leftmost derivations of CFGs to logic programs.

Magic predicates

$$1\colon S^{bf}(i,m) :- P^{bfff}(i,j,k,l), Q^{bbbf}(j,k,l,m).$$
$$2\colon P^{bfff}(i,j,k,l) :- a^{bf}(i,j), c^{ff}(k,l).$$
$$3\colon P^{bfff}(i,k,l,n) :- a^{bf}(i,j), P^{bfff}(j,k,m,n), c^{fb}(l,m).$$
$$4\colon Q^{bbbf}(i,j,k,l) :- b^{bb}(i,j), d^{bf}(k,l).$$
$$5\colon Q^{bbbf}(i,k,l,n) :- b^{bf}(i,j), d^{bf}(l,m), Q^{bbbf}(j,k,m,n).$$

$$?- S(0,x). \qquad m\_S(0)$$
$$\stackrel{1}{\Longrightarrow} ?- P(0,j_1,k_1,l_1), Q(j_1,k_1,l_1,x). \qquad m\_P(0)$$
$$\stackrel{2}{\Longrightarrow} ?- a(0,j_1), c(k_1,l_1), Q(j_1,k_1,l_1,x).$$
$$\stackrel{a(0,1)}{\Longrightarrow} ?- c(k_1,l_1), Q(1,k_1,l_1,x).$$
$$\stackrel{c(2,3)}{\Longrightarrow} ?- Q(1,2,3,x). \qquad m\_Q(1,2,3)$$
$$\stackrel{4}{\Longrightarrow} ?- b(1,2), d(3,x).$$
$$\stackrel{b(1,2)}{\Longrightarrow} ?- d(3,x).$$
$$\stackrel{d(3,4)}{\Longrightarrow} ?- [].$$

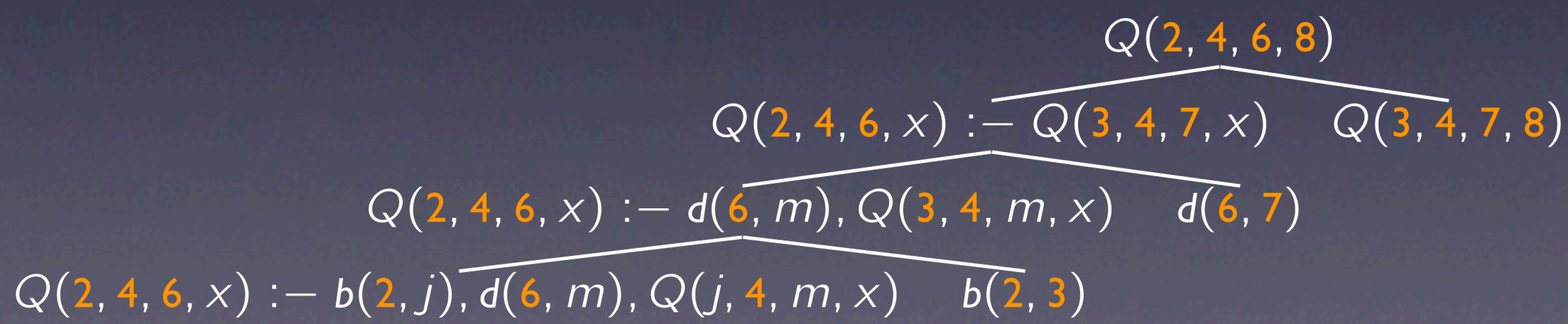
The fact that S(0,x) is the first goal is represented by the “magic” predicate m_S(0), etc.
Magic predicates take only bound arguments.

Supplementary predicates

$$5\colon Q^{bbbf}(i,k,l,n) :- b^{bf}(i,j), d^{bf}(l,m), Q^{bbbf}(j,k,m,n).$$

SLD derivation

$$?- Q(2,4,6,x).$$
$$\xRightarrow{5} ?- b(2,j), d(6,m), Q(j,4,m,x).$$
$$\xRightarrow{b(2,3)} ?- d(6,m), Q(3,4,m,x).$$
$$\xRightarrow{d(6,7)} ?- Q(3,4,7,x).$$
$$\vdots$$
$$\Longrightarrow ?- [].$$



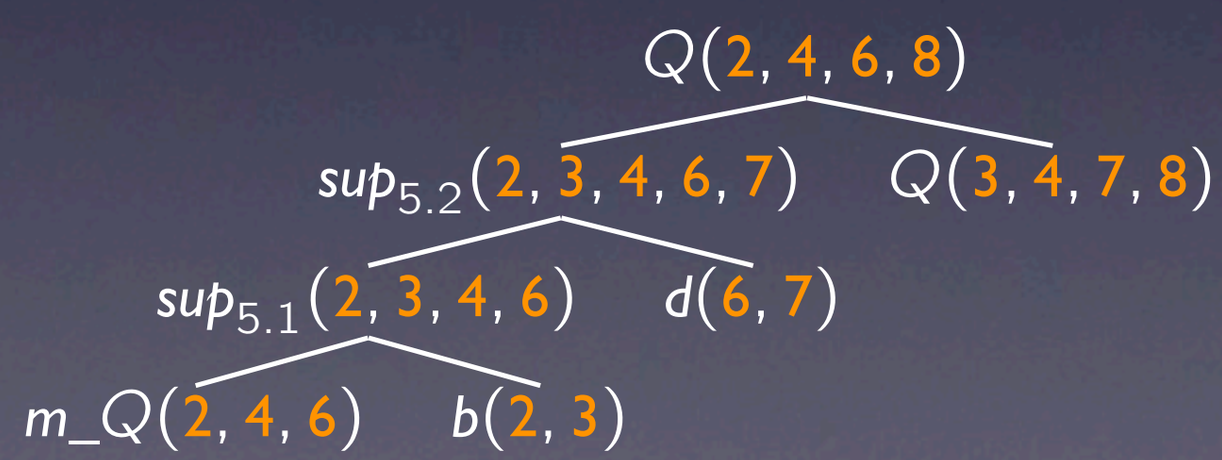
Here's how rule 5 is converted to Datalog rules.
A Datalog derivation corresponding to an SLD derivation.

Supplementary predicates

$$5\colon Q^{bbbf}(i,k,l,n) :- b^{bf}(i,j), d^{bf}(l,m), Q^{bbbf}(j,k,m,n).$$

SLD derivation

$$?- Q(2,4,6,x).$$
$$\xRightarrow{5} ?- b(2,j), d(6,m), Q(j,4,m,x).$$
$$\xRightarrow{b(2,3)} ?- d(6,m), Q(3,4,m,x).$$
$$\xRightarrow{d(6,7)} ?- Q(3,4,7,x).$$
$$\vdots$$
$$\Longrightarrow ?- [].$$

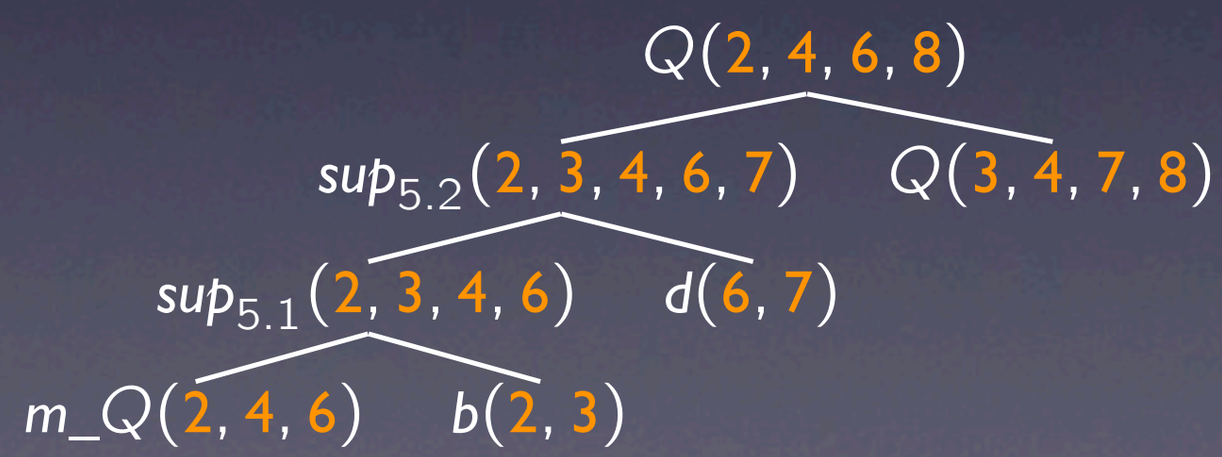


A partially instantiated clause is represented with a magic predicate.
A derived clause is represented with a supplementary predicate.

Rules for supplementary predicates

$$5\colon Q^{bbbf}(i,k,l,n) :- b^{bf}(i,j), d^{bf}(l,m), Q^{bbbf}(j,k,m,n).$$

$$sup_{5.1}(i,j,k,l) :- m\_Q(i,k,l), b(i,j).$$
$$sup_{5.2}(i,j,k,l,m) :- sup_{5.1}(i,j,k,l), d(l,m).$$
$$Q(i,k,l,n) :- sup_{5.2}(i,j,k,l,m), Q(j,k,m,n).$$



Rules for magic predicates

1: $S^{bf}(i, m) :- P^{bfff}(i, j, k, l), Q^{bbbf}(j, k, l, m).$

2: $P^{bfff}(i, j, k, l) :- a^{bf}(i, j), c^{ff}(k, l).$

3: $P^{bfff}(i, k, l, n) :- a^{bf}(i, j), P^{bfff}(j, k, m, n), c^{fb}(l, m).$

4: $Q^{bbbf}(i, j, k, l) :- b^{bb}(i, j), d^{bf}(k, l).$

5: $Q^{bbbf}(i, k, l, n) :- b^{bf}(i, j), d^{bf}(l, m), Q^{bbbf}(j, k, m, n).$

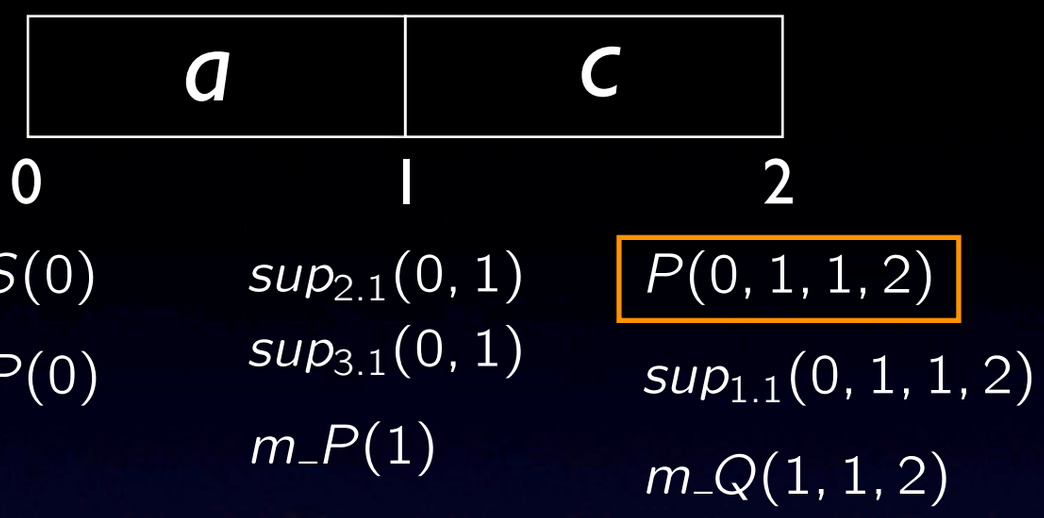
$m\_P(i) :- m\_S(i).$

$m\_Q(j, k, l) :- sup_{1.1}(i, j, k, l).$

$m\_P(j) :- sup_{3.1}(i, j).$

$m\_Q(j, k, m) :- sup_{5.2}(i, j, k, l, m).$

$r_1 : m\_P(i) :- m\_S(i).$
$r_2 : m\_Q(j,k,l) :- sup_{1.1}(i,j,k,l).$
$r_3 : m\_P(j) :- sup_{3.1}(i,j).$
$r_4 : m\_Q(j,k,m) :- sup_{5.2}(i,j,k,l,m).$
$r_5 : sup_{1.1}(i,j,k,l) :- m\_S(i), P(i,j,k,l).$
$r_6 : S(i,m) :- sup_{1.1}(i,j,k,l), Q(j,k,l,m).$
$r_7 : sup_{2.1}(i,j) :- m\_P(i), a(i,j).$
$\boxed{r_8 : P(i,j,k,l) :- sup_{2.1}(i,j), c(k,l).}$
$r_9 : sup_{3.1}(i,j) :- m\_P(i), a(i,j).$
$r_{10} : sup_{3.2}(i,k,m,n) :- sup_{3.1}(i,j), P(j,k,m,n).$
$r_{11} : P(i,k,l,n) :- sup_{3.2}(i,k,m,n), c(l,m).$
$r_{12} : sup_{4.1}(i,j,k) :- m\_Q(i,j,k), b(i,j).$
$r_{13} : Q(i,j,k,l) :- sup_{4.1}(i,j,k), d(k,l).$
$r_{14} : sup_{5.1}(i,j,k,l) :- m\_Q(i,k,l), b(i,j).$
$r_{15} : sup_{5.2}(i,j,k,l,m) :- sup_{5.1}(i,j,k,l), d(l,m).$
$r_{16} : Q(i,k,l,n) :- sup_{5.2}(i,j,k,l,m), Q(j,k,m,n).$



not prefix-correct!

Magic-sets corresponds to Earley for CFGs, but in general, does not imply prefix-correctness.

Cause of non-prefix-correctness

SLD derivation

$$?- S(0, x).$$

$$\xRightarrow{1} ?- P(0, j_1, k_1, l_1), Q(j_1, k_1, l_1, x).$$

$$\xRightarrow{2} ?- a(0, j_1), c(k_1, l_1), Q(j_1, k_1, l_1, x).$$

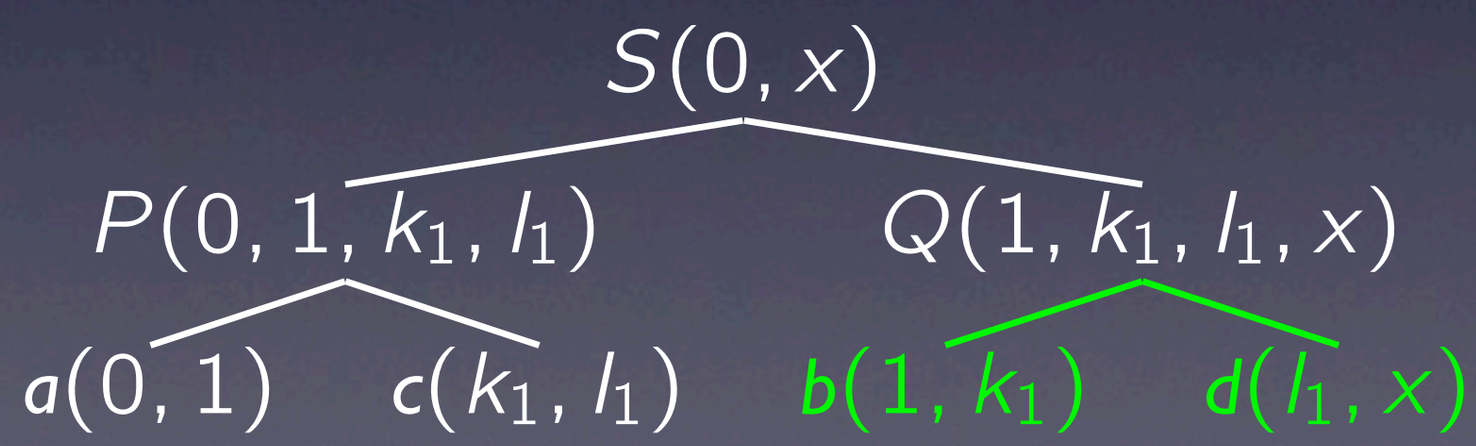
$$\xRightarrow{a(0,1)} ?- c(k_1, l_1), Q(1, k_1, l_1, x).$$

$$\xRightarrow{c(1,2)} ?- Q(1, 1, 2, x).$$

$$\xRightarrow{4} ?- b(1, 1), d(2, x).$$

adornment

$$2\colon P^{bfff}(i, j, k, l) :- a^{bf}(i, j), c^{ff}(k, l).$$



incomplete derivation tree

SLD derivation with the original program is not prefix-correct.
The order of extensional facts at the leaves of the derivation tree does not match the order of terminals in the input.

Securing prefix-correctness

$$S(\ x_1\ y_1\ x_2\ y_2\ ) :- P(x_1, x_2), Q(y_1, y_2).$$
$$\quad i \quad j \quad k \quad l \quad m$$

$$\cancel{S(i, m) :- P(i, j, k, l), Q(j, k, l, m).}$$

$$S(i, m) :- P_1(i, j), Q_1(j, k), P(i, j, k, l), Q(j, k, l, m).$$

$$\cancel{P(i, j, k, l) :- a(i, j), c(k, l).}$$
$$\cancel{P_1(i, j) :- a(i, j).}$$

$$P(i, j, k, l) :- aux(i, j), c(k, l).$$
$$P_1(i, j) :- aux(i, j).$$
$$aux(i, j) :- a(i, j).$$

$$S^{bf}(i, m) :- P_1^{bf}(i, j), Q_1^{bf}(j, k), P^{bbbf}(i, j, k, l), Q^{bbbf}(j, k, l, m).$$
$$P^{bbbf}(i, j, k, l) :- aux^{bb}(i, j), c^{bf}(k, l).$$
$$P_1^{bf}(i, j) :- aux^{bf}(i, j).$$
$$aux^{bf}(i, j) :- a^{bf}(i, j).$$

Add “redundant” subgoals.
The aux predicate “folds” common part of two rules.

New starting point

1: $S^{bf}(i, m) :- P_1^{bf}(i, j), Q_1^{bf}(j, k), P^{bbbf}(i, j, k, l), Q^{bbbf}(j, k, l, m).$

2: $P_1^{bf}(i, j) :- aux_2^{bf}(i, j).$

3: $P^{bbbf}(i, j, k, l) :- aux_2^{bf}(i, j), c^{bf}(k, l).$

4: $aux_2^{bf}(i, j) :- a^{bf}(i, j).$

5: $P_1^{bf}(i, k) :- aux_3^{bff}(i, j, k).$

6: $P^{bbbf}(i, k, l, n) :- aux_3^{bff}(i, j, k), c^{bf}(l, m), P^{bbbf}(j, k, m, n).$

7: $aux_3^{bff}(i, j, k) :- a^{bf}(i, j), P_1^{bf}(j, k).$

8: $Q_1^{bf}(i, j) :- aux_4^{bf}(i, j).$

9: $Q^{bbbf}(i, j, k, l) :- aux_4^{bf}(i, j), d^{bf}(k, l).$

10: $aux_4^{bf}(i, j) :- b^{bf}(i, j).$

11: $Q_1^{bf}(i, k) :- aux_5^{bff}(i, j, k).$

12: $Q^{bbbf}(i, k, l, n) :- aux_5^{bff}(i, j, k), d^{bf}(l, m), Q^{bbbf}(j, k, m, n).$

13: $aux_5^{bff}(i, j, k) :- b^{bf}(i, j), Q_1^{bf}(j, k).$

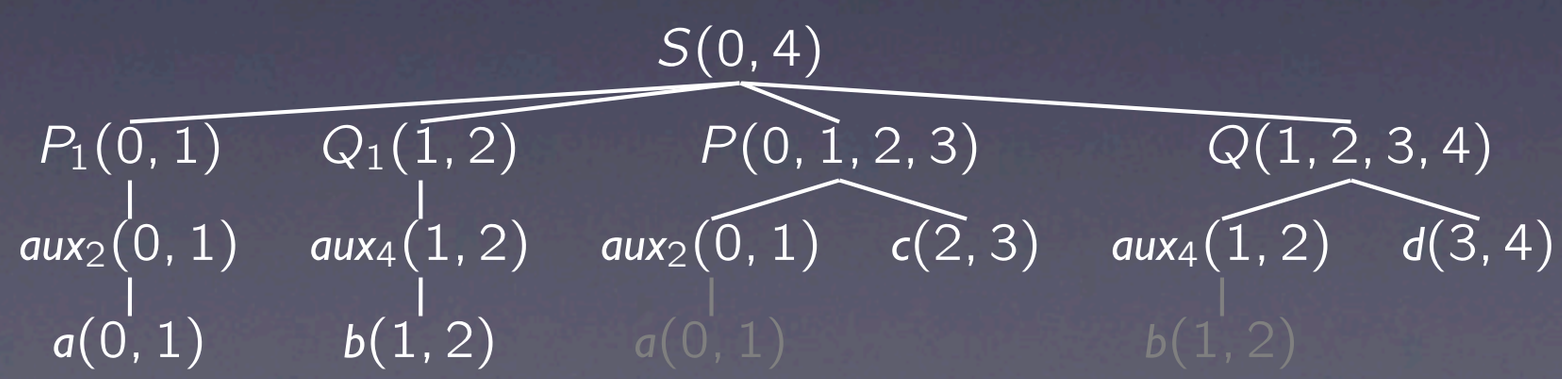
The result of introducing redundant subgoals to the original program.

Prefix-correctness

SLD derivation

$$
\begin{aligned}
& ?- S(0,x). \\
\overset{1}{\Longrightarrow}\ & ?- P_1(0,j_1), Q_1(j_1,k_1), P(0,j_1,k_1,l_1), Q(j_1,k_1,l_1,x). \\
\overset{2}{\Longrightarrow}\ & ?- aux_2(0,j_i), Q_1(j_1,k_1), P(0,j_1,k_1,l_1), Q(j_1,k_1,l_1,x). \\
\overset{4}{\Longrightarrow}\ & ?- a(0,j_1), Q_1(j_1,k_1), P(0,j_1,k_1,l_1), Q(j_1,k_1,l_1,x). \\
\overset{a(0,1)}{\Longrightarrow}\ & ?- Q_1(1,k_1), P(0,1,k_1,l_1), Q(1,k_1,l_1,x). \\
\overset{8}{\Longrightarrow}\ & ?- aux_4(1,k_1), P(0,1,k_1,l_1), Q(1,k_1,l_1,x). \\
\overset{10}{\Longrightarrow}\ & ?- b(1,k_1), P(0,j_1,k_1,l_1), Q(1,k_1,l_1,x). \\
\overset{b(1,2)}{\Longrightarrow}\ & ?- P(0,1,2,l_1), Q(1,2,l_1,x). \\
\overset{3}{\Longrightarrow}\ & ?- aux_3(0,1), c(2,l_1), Q(1,2,l_1,x). \\
& \vdots \\
\Longrightarrow\ & ?- c(2,l_1), Q(1,2,l_1,x). \\
\overset{c(2,3)}{\Longrightarrow}\ & ?- Q(1,2,3,x). \\
\overset{9}{\Longrightarrow}\ & ?- aux_4(1,2), d(3,x). \\
& \vdots \\
\Longrightarrow\ & ?- d(3,x). \\
\overset{d(3,4)}{\Longrightarrow}\ & ?- []
\end{aligned}
$$

derivation tree



SLD derivation is now prefix-correct.

Magic-sets rewriting

r_1: $m\_P_1(i) :- m\_S(i).$
r_2: $m\_Q_1(j) :- sup_{1.1}(i,j).$
r_3: $m\_P(i,j,k) :- sup_{1.2}(i,j,k).$
r_4: $m\_Q(j,k,l) :- sup_{1.3}(i,j,k,l).$
r_5: $m\_aux_2(i) :- m\_P_1(i).$
r_6: $m\_aux_2(i) :- m\_P(i,j,k).$
r_7: $m\_aux_3(j) :- m\_P_1(i).$
r_8: $m\_aux_3(i) :- m\_P(i,k,l).$
r_9: $m\_P(j,k,m) :- sup_{6.2}(i,j,k,l,m).$
r_{10}: $m\_P_1(j) :- sup_{7.1}(i,j).$
r_{11}: $m\_aux_4(i) :- m\_Q_1(i).$
r_{12}: $m\_aux_4(i) :- m\_Q(i,j,k).$
r_{13}: $m\_aux_5(i) :- m\_Q_1(i).$
r_{14}: $m\_aux_5(i) :- m\_Q(i,k,l).$
r_{15}: $m\_Q(j,k,m) :- sup_{12.2}(i,j,k,l,m).$
r_{16}: $m\_Q_1(j) :- sup_{13.1}(i,j).$

r_{17}: $sup_{1.1}(i,j) :- m\_S(i), P_1(i,j).$
r_{18}: $sup_{1.2}(i,j,k) :- sup_{1.1}(i,j), Q_1(j,k).$
r_{19}: $sup_{1.3}(i,j,k,l) :- sup_{1.2}(i,j,k), P(i,j,k,l).$
r_{20}: $S(i,m) :- sup_{1.3}(i,j,k,l), Q(j,k,l,m).$

r_{21}: $P_1(i,j) :- m\_P_1(i), aux_2(i,j).$
r_{22}: $sup_{3.1}(i,j,k) :- m\_P(i,j,k), aux_2(i,j).$
r_{23}: $P(i,j,k,l) :- sup_{3.1}(i,j,k), c(k,l).$
r_{24}: $aux_2(i,j) :- m\_aux_2(i), a(i,j).$
r_{25}: $P_1(i,k) :- m\_P_1(i), aux_3(i,j,k).$
r_{26}: $sup_{6.1}(i,j,k,l) :- m\_P(i,k,l), aux_3(i,j,k).$
r_{27}: $sup_{6.2}(i,j,k,l,m) :- sup_{6.1}(i,j,k,l), c(l,m).$
r_{28}: $P(i,k,l,n) :- sup_{6.2}(i,j,k,l,m), P(j,k,m,n).$
r_{29}: $sup_{7.1}(i,j) :- m\_aux_3(i), a(i,j).$
r_{30}: $aux_3(i,j,k) :- sup_{7.1}(i,j), P_1(j,k).$
r_{31}: $Q_1(i,j) :- m\_Q_1(i), aux_4(i,j).$
r_{32}: $sup_{9.1}(i,j,k) :- m\_Q(i,j,k), aux_4(i,j).$
r_{33}: $Q(i,j,k,l) :- sup_{9.1}(i,j,k), d(k,l).$
r_{34}: $aux_4(i,j) :- m\_aux_4(i), b(i,j).$
r_{35}: $Q_1(i,k) :- m\_Q_1(i), aux_5(i,_j, k).$
r_{36}: $sup_{12.1}(i,j,k,l) :- m\_Q(i,k,l), aux_5(i,j,k).$
r_{37}: $sup_{12.2}(i,j,k,l,m) :- sup_{12.1}(i,j,k,l), d(l,m).$
r_{38}: $Q(i,k,l,n) :- sup_{12.2}(i,j,k,l,m), Q(j,k,m,n).$
r_{39}: $sup_{13.1}(i,j) :- m\_aux_5(i), b(i,j).$
r_{40}: $aux_5(i,j,k) :- sup_{13.1}(i,j), Q_1(j,k).$

How parsing should be approached

tabular parsing

parsing schema	control algorithm
program transformation	canonical chart parsing (≈ seminaive bottom-up evaluation)
(grammar → deduction system)	
Datalog program	Datalog program

- Use well-established, general formalism/method
- Avoids ad hoc techniques as much as possible
- Easier to understand and easier to prove correct

Some people even divide parsing schema into conversion to PDA and “PDA tabulation”. This is not necessarily a natural approach.

How parsing should be approached

Earley parsing

parsing schema	control algorithm
introduction of redundant predicates + magic-sets rewriting	canonical chart parsing (≈ seminaive bottom-up evaluation)
(grammar → deduction system)	
Datalog program	Datalog program

- Use well-established, general formalism/method
- Avoids ad hoc techniques as much as possible
- Easier to understand and easier to prove correct

In the case of Earley, introduction of redundant predicates plus magic-sets rewriting.