Parsing of Context-Free Grammars

Bernd Kiefer
Bernd.Kiefer@dfki.de

Deutsches Forschungszentrum für künstliche Intelligenz

October 25, 2010

Limits of Regular Expressions

- Assignment: Write a regular expression for fully bracketed arithmetic expressions!

Limits of Regular Expressions

- Assignment: Write a regular expression for fully bracketed arithmetic expressions!
- Answer:
  - This is not possible!

Limits of Regular Expressions

- Assignment: Write a regular expression for fully bracketed arithmetic expressions!
- Answer:
  - This is not possible!
  - Regular expressions can only count *finite* amounts of brackets
  - We need a more powerful formal device: *context-free grammars*

Limits of Regular Expressions

- Assignment: Write a regular expression for fully bracketed arithmetic expressions!
- Answer:
  - This is not possible!
  - Regular expressions can only count *finite* amounts of brackets
  - We need a more powerful formal device: *context-free grammars*
  - Context-free grammars provide a (finite) inventory of named brackets
  - All regular languages are also context-free, i.e.:
    for every regular expression, there is a context-free grammar that accepts / derives the same language

Context-Free Grammars

A context-free grammar (CFG) consists of:

- The set of terminal symbols $\Sigma = a, b, c, \ldots$ (the words or letters of the language)
- The set of non-terminal symbols $N = A, B, C, \ldots$
- The startsymbol $S \in N$
- The set of productions (rules) P, where
  $P \ni r = A \rightarrow \alpha$ with $\alpha \in (\Sigma \cup N)^*$
  (we use greek letters for strings of $\Sigma \cup N$)

Example: A grammar for arithmetic expressions:

$$\Sigma = \{ \texttt{int}, \texttt{+}, \texttt{*}, \texttt{(}, \texttt{)} \} \quad , \quad N = \{E\} \quad , \quad S = E$$
$$P = \{ \quad E \rightarrow E\texttt{+}E, \quad E \rightarrow E\,\texttt{*}E, \quad E \rightarrow \texttt{(}E\texttt{)}, \quad E \rightarrow \texttt{int} \quad \}$$

The Language of a CFG

- Given a CFG G, the language $\mathcal{L}(G)$ is defined as the set of all strings that can be *derived* from S
- Given a string α from $(\Sigma \cup N)^*$, derive a new string β:
  - Choose one of the nonterminals in α, say, A
  - Choose one of the productions with A on the left hand side
  - Replace A in α with the right hand side (rhs) of the production to get the derived string β
- If α contains only symbols in Σ, then $\alpha \in \mathcal{L}(G)$
- Example:

  $\alpha =$ `int*`(E); choose $E \rightarrow E$ `+`E; $\beta =$ `int*(`E`+`E`)`

Derivations: Formally

- A string α derives a string β, $(\alpha \underset{G}{\Rightarrow} \beta)$ $\alpha, \beta \in (\Sigma \cup N)^*$, if:
  there are $\gamma, \delta, \eta \in (\Sigma \cup N)^*, A \in N$ such that
  $\alpha = \gamma A \delta \quad , \quad \beta = \gamma \eta \delta \quad$ and $\quad A \longrightarrow \eta \in P$
- We write $\alpha \underset{G}{\Rightarrow} \beta$ for a one-step derivation
- $\alpha \overset{*}{\underset{G}{\Rightarrow}} \beta$ is a many-step derivation: $\alpha \underset{G}{\Rightarrow} \alpha_0 \underset{G}{\Rightarrow} \alpha_1 \ldots \underset{G}{\Rightarrow} \beta$
- Language $\mathcal{L}(G)$ generated by G: $\mathcal{L}(G) = \{s \in \Sigma^* | S \overset{*}{\underset{G}{\Rightarrow}} s\}$
- The task of a parser: find one (or all) derivation(s) of a string in Σ^*, given a CFG G

Derivations: Example

- $\Sigma = \{\textit{john, girl, car, saw, walks, in, the, a}\}$
- $N = \{\text{S, NP, VP, PP, D, N, V, P}\}$
- $P = \left\{ \begin{array}{ll} \text{S} \rightarrow \text{NP VP} \mid \text{N VP} \mid \text{N V} \mid \text{NP V} & \text{N} \rightarrow \textit{john, girl, car} \\ \text{VP} \rightarrow \text{V NP} \mid \text{V N} \mid \text{VP PP} & \text{V} \rightarrow \textit{saw, walks} \\ \text{NP} \rightarrow \text{D N} \mid \text{NP PP} \mid \text{N PP} & \text{P} \rightarrow \textit{in} \\ \text{PP} \rightarrow \text{P NP} \mid \text{P N} & \text{D} \rightarrow \textit{the, a} \end{array} \right\}$

$\text{S} \underset{G}{\Rightarrow}$

john saw the girl in a car

Derivations: Example

- $\Sigma = \{john, girl, car, saw, walks, in, the, a\}$
- $N = \{\text{S, NP, VP, PP, D, N, V, P}\}$
- $P = \left\{ \begin{array}{ll} \text{S} \rightarrow \text{NP VP} | \text{N VP} | \text{N V} | \text{NP V} & \text{N} \rightarrow john, girl, car \\ \text{VP} \rightarrow \text{V NP} | \text{V N} | \text{VP PP} & \text{V} \rightarrow saw, walks \\ \text{NP} \rightarrow \text{D N} | \text{NP PP} | \text{N PP} & \text{P} \rightarrow in \\ \text{PP} \rightarrow \text{P NP} | \text{P N} & \text{D} \rightarrow the, a \end{array} \right\}$

$\text{S} \underset{G}{\Rightarrow} \text{N VP} \underset{G}{\Rightarrow}$

john saw the girl in a car

Derivations: Example

- $\Sigma = \{john, girl, car, saw, walks, in, the, a\}$
- $N = \{\text{S, NP, VP, PP, D, N, V, P}\}$
- $P = \left\{ \begin{array}{ll} \text{S} \rightarrow \text{NP VP} | \text{N VP} | \text{N V} | \text{NP V} & \text{N} \rightarrow john, girl, car \\ \text{VP} \rightarrow \text{V NP} | \text{V N} | \text{VP PP} & \text{V} \rightarrow saw, walks \\ \text{NP} \rightarrow \text{D N} | \text{NP PP} | \text{N PP} & \text{P} \rightarrow in \\ \text{PP} \rightarrow \text{P NP} | \text{P N} & \text{D} \rightarrow the, a \end{array} \right\}$

$$\text{S} \underset{G}{\Rightarrow} \text{N VP} \underset{G}{\Rightarrow} \text{john VP} \underset{G}{\Rightarrow}$$

john saw the girl in a car

Derivations: Example

- $\Sigma = \{john, girl, car, saw, walks, in, the, a\}$
- $N = \{\text{S, NP, VP, PP, D, N, V, P}\}$
- $P = \left\{ \begin{array}{ll} \text{S} \rightarrow \text{NP VP} | \text{N VP} | \text{N V} | \text{NP V} & \text{N} \rightarrow john, girl, car \\ \text{VP} \rightarrow \text{V NP} | \text{V N} | \text{VP PP} & \text{V} \rightarrow saw, walks \\ \text{NP} \rightarrow \text{D N} | \text{NP PP} | \text{N PP} & \text{P} \rightarrow in \\ \text{PP} \rightarrow \text{P NP} | \text{P N} & \text{D} \rightarrow the, a \end{array} \right\}$

$$\text{S} \underset{G}{\Rightarrow} \text{N VP} \underset{G}{\Rightarrow} \text{john VP} \underset{G}{\Rightarrow} \text{john V NP} \underset{G}{\Rightarrow}$$

john saw the girl in a car

Derivations: Example

- $\Sigma = \{john, girl, car, saw, walks, in, the, a\}$
- $N = \{\text{S, NP, VP, PP, D, N, V, P}\}$
- $P = \left\{ \begin{array}{ll} \text{S} \rightarrow \text{NP VP} | \text{N VP} | \text{N V} | \text{NP V} & \text{N} \rightarrow john, girl, car \\ \text{VP} \rightarrow \text{V NP} | \text{V N} | \text{VP PP} & \text{V} \rightarrow saw, walks \\ \text{NP} \rightarrow \text{D N} | \text{NP PP} | \text{N PP} & \text{P} \rightarrow in \\ \text{PP} \rightarrow \text{P NP} | \text{P N} & \text{D} \rightarrow the, a \end{array} \right\}$

$$\text{S} \underset{G}{\Rightarrow} \text{N VP} \underset{G}{\Rightarrow} \text{john VP} \underset{G}{\Rightarrow} \text{john V NP} \underset{G}{\Rightarrow} \text{john saw NP} \underset{G}{\Rightarrow}$$

john saw the girl in a car

Derivations: Example

- $\Sigma = \{john, girl, car, saw, walks, in, the, a\}$
- $N = \{\text{S, NP, VP, PP, D, N, V, P}\}$
- $P = \left\{ \begin{array}{ll} \text{S} \rightarrow \text{NP VP} | \text{N VP} | \text{N V} | \text{NP V} & \text{N} \rightarrow john, girl, car \\ \text{VP} \rightarrow \text{V NP} | \text{V N} | \text{VP PP} & \text{V} \rightarrow saw, walks \\ \text{NP} \rightarrow \text{D N} | \text{NP PP} | \text{N PP} & \text{P} \rightarrow in \\ \text{PP} \rightarrow \text{P NP} | \text{P N} & \text{D} \rightarrow the, a \end{array} \right\}$

$\text{S} \underset{G}{\Rightarrow} \text{N VP} \underset{G}{\Rightarrow} \text{john VP} \underset{G}{\Rightarrow} \text{john V NP} \underset{G}{\Rightarrow} \text{john saw NP} \underset{G}{\Rightarrow}$
$\text{john saw NP PP} \underset{G}{\Rightarrow}$

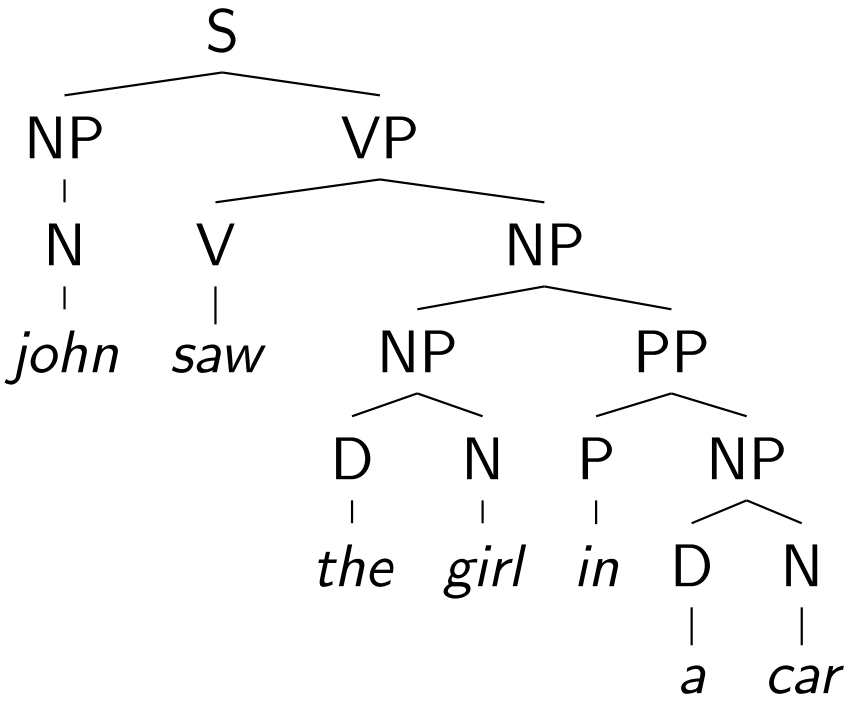
john saw the girl in a car

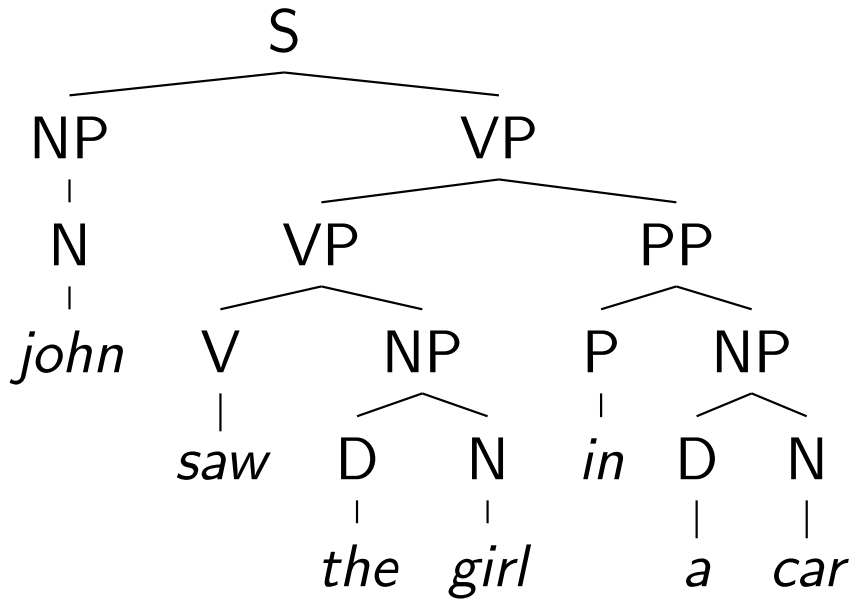
Derivations: Example

- $\Sigma = \{john, girl, car, saw, walks, in, the, a\}$
- $N = \{\mathrm{S}, \mathrm{NP}, \mathrm{VP}, \mathrm{PP}, \mathrm{D}, \mathrm{N}, \mathrm{V}, \mathrm{P}\}$
- $P = \left\{ \begin{array}{ll} \mathrm{S} \rightarrow \mathrm{NP\ VP} | \mathrm{N\ VP} | \mathrm{N\ V} | \mathrm{NP\ V} & \mathrm{N} \rightarrow john, girl, car \\ \mathrm{VP} \rightarrow \mathrm{V\ NP} | \mathrm{V\ N} | \mathrm{VP\ PP} & \mathrm{V} \rightarrow saw, walks \\ \mathrm{NP} \rightarrow \mathrm{D\ N} | \mathrm{NP\ PP} | \mathrm{N\ PP} & \mathrm{P} \rightarrow in \\ \mathrm{PP} \rightarrow \mathrm{P\ NP} | \mathrm{P\ N} & \mathrm{D} \rightarrow the, a \end{array} \right\}$

$$\mathrm{S} \underset{G}{\Rightarrow} \mathrm{N\ VP} \underset{G}{\Rightarrow} \text{john VP} \underset{G}{\Rightarrow} \text{john V NP} \underset{G}{\Rightarrow} \text{john saw NP} \underset{G}{\Rightarrow}$$
$$\text{john saw NP PP} \underset{G}{\Rightarrow} \text{john saw D N PP} \underset{G}{\Rightarrow} \text{john saw the N PP} \underset{G}{\Rightarrow}$$
$$\text{john saw the girl PP} \underset{G}{\Rightarrow} \text{john saw the girl P NP} \underset{G}{\Rightarrow}$$
$$\text{john saw the girl in NP} \underset{G}{\Rightarrow} \text{john saw the girl in D N} \underset{G}{\Rightarrow}$$
$$\text{john saw the girl in a N} \underset{G}{\Rightarrow}$$
$$\text{john saw the girl in a car}$$

Derivation (Parse) Trees





- Encodes many possible derivations
- PP node in the example can be attached to two nodes: the grammar is ambigous
- CF Parsers/Recognizers differ in the way the derivation trees are build

Context-free Recognition

Task: given $s \in \Sigma^*$ and G, is $s \in \mathcal{L}(G)$?

Two ways to go:

- start with the start symbol S and try to derive s by systematic application of the productions:
  top down recognition (goal driven)
- start with the string s and try to reduce it to the start symbol:
  bottom up recognition (data driven)

Recursive Descent Parsing

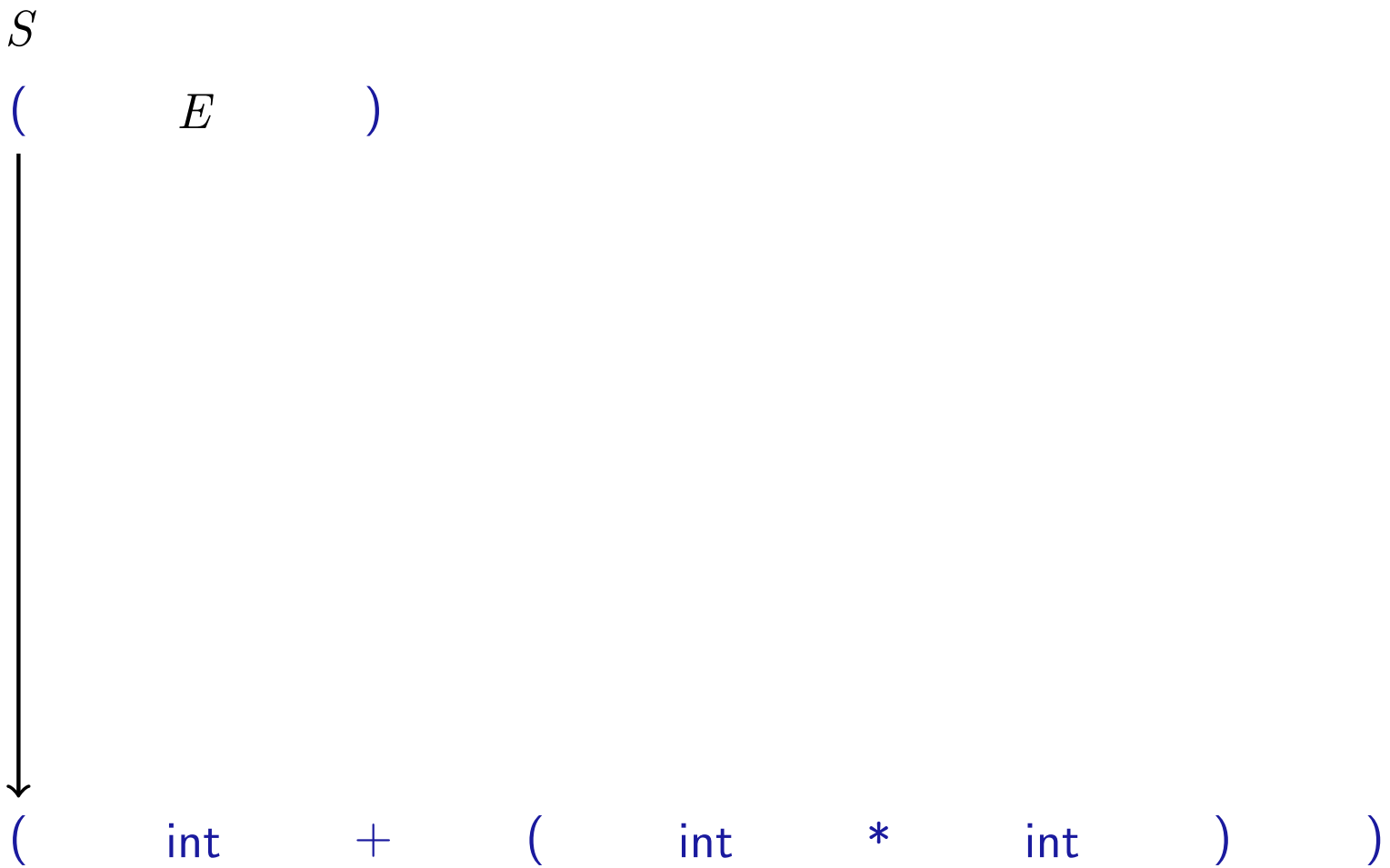
Idea: Recursively compute all expansions of a nonterminal at some input position

S

(int + (int * int))

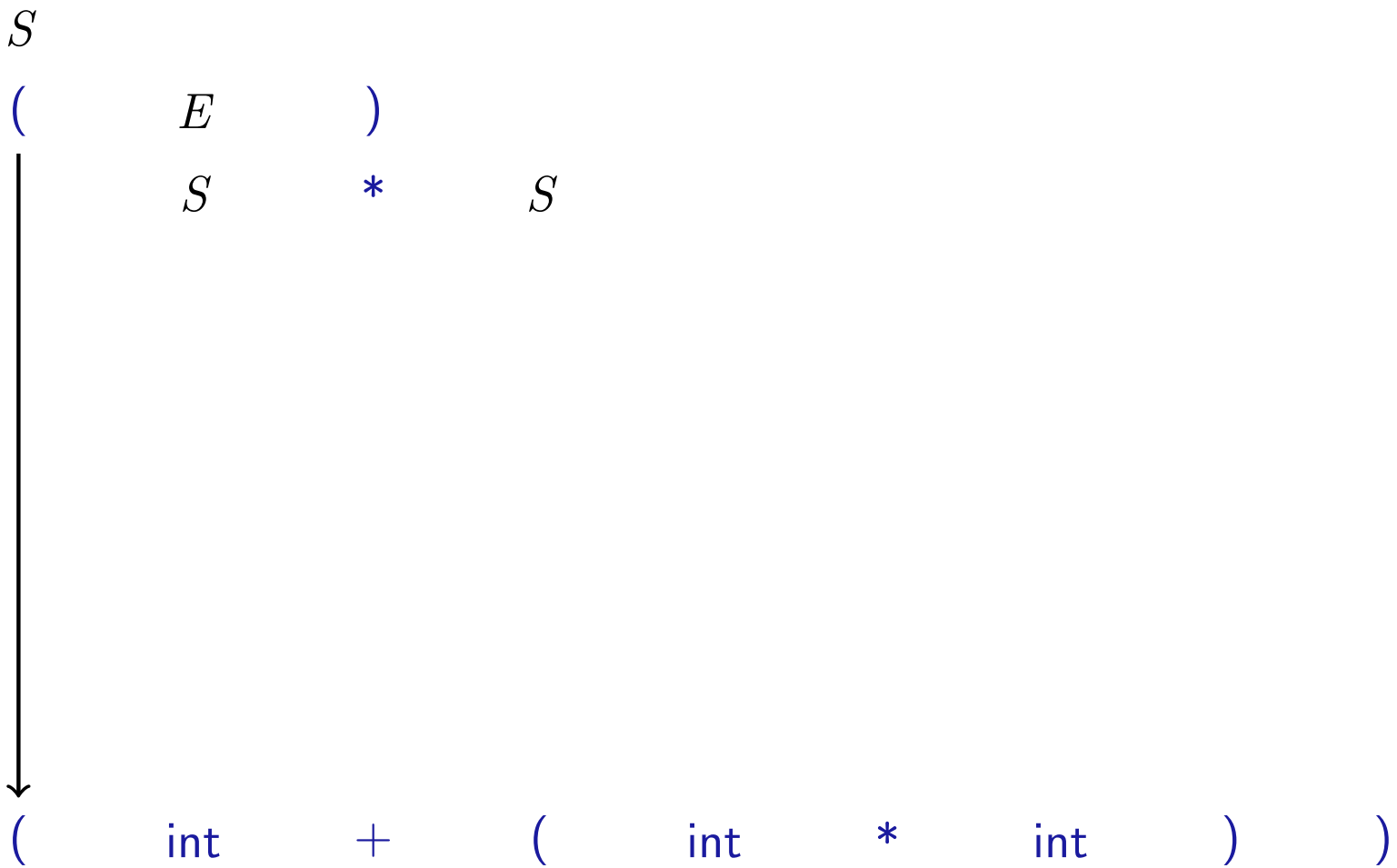
Recursive Descent Parsing

Idea: Recursively compute all expansions of a nonterminal at some input position



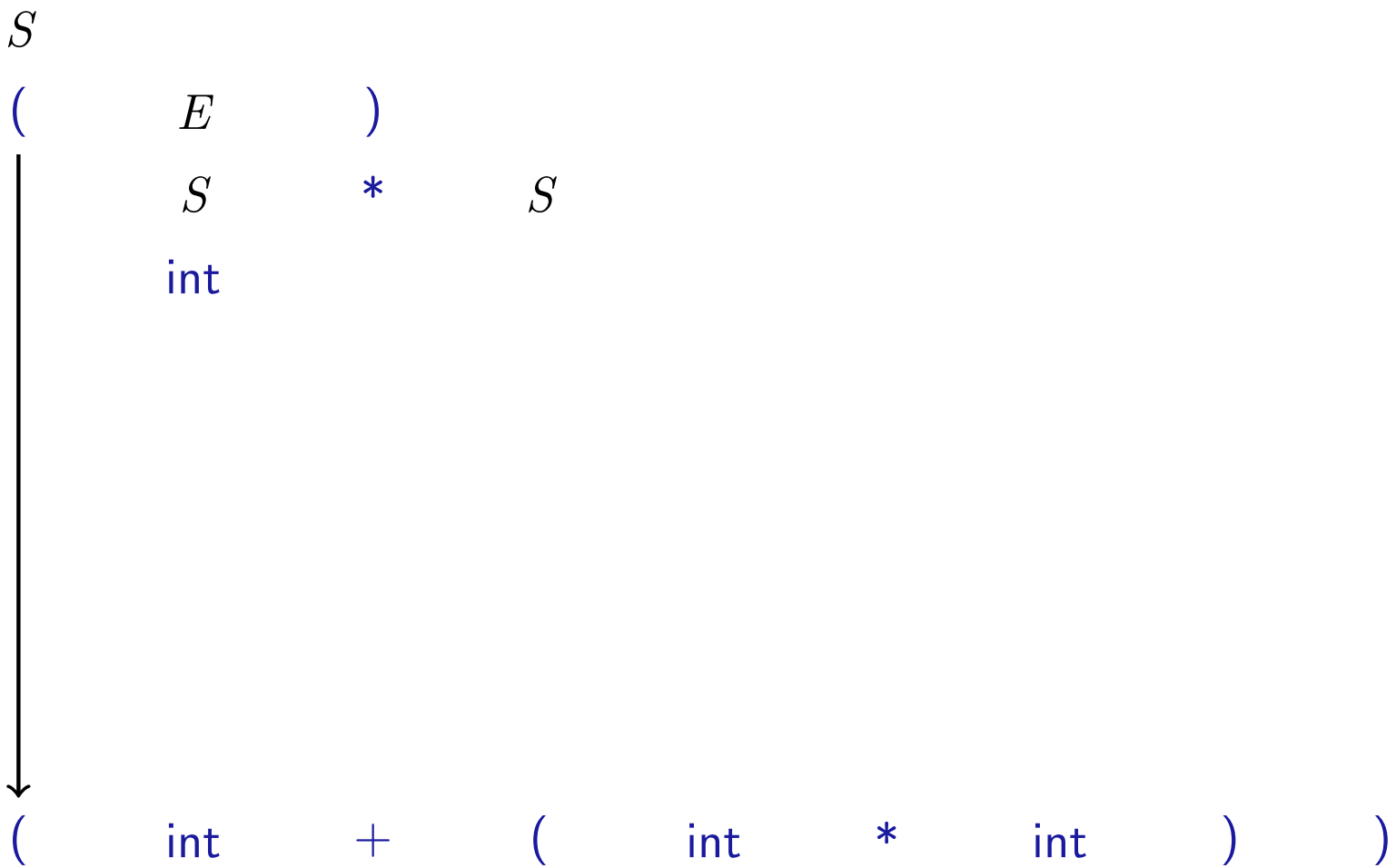
Recursive Descent Parsing

Idea: Recursively compute all expansions of a nonterminal at some input position



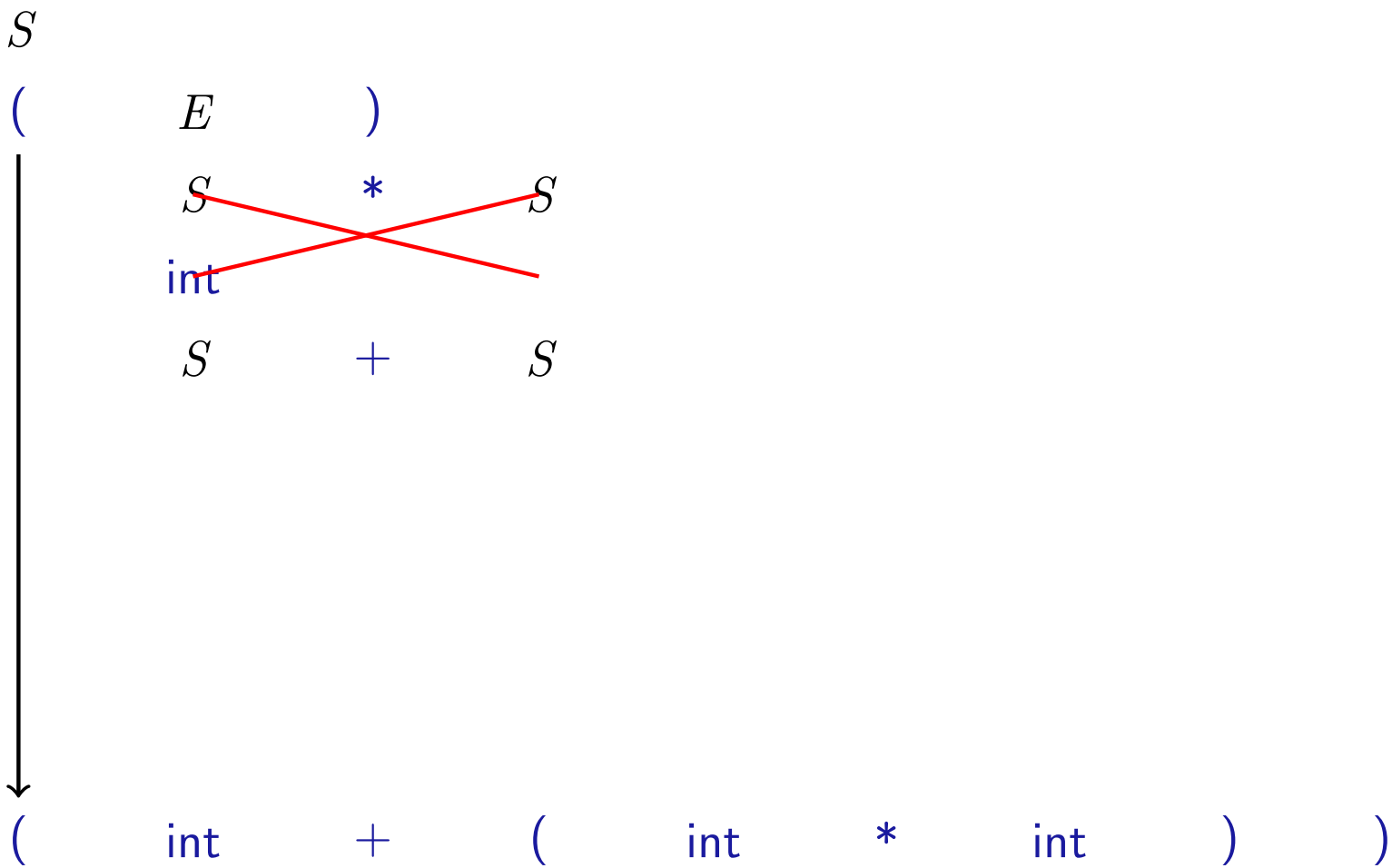
Recursive Descent Parsing

Idea: Recursively compute all expansions of a nonterminal at some input position



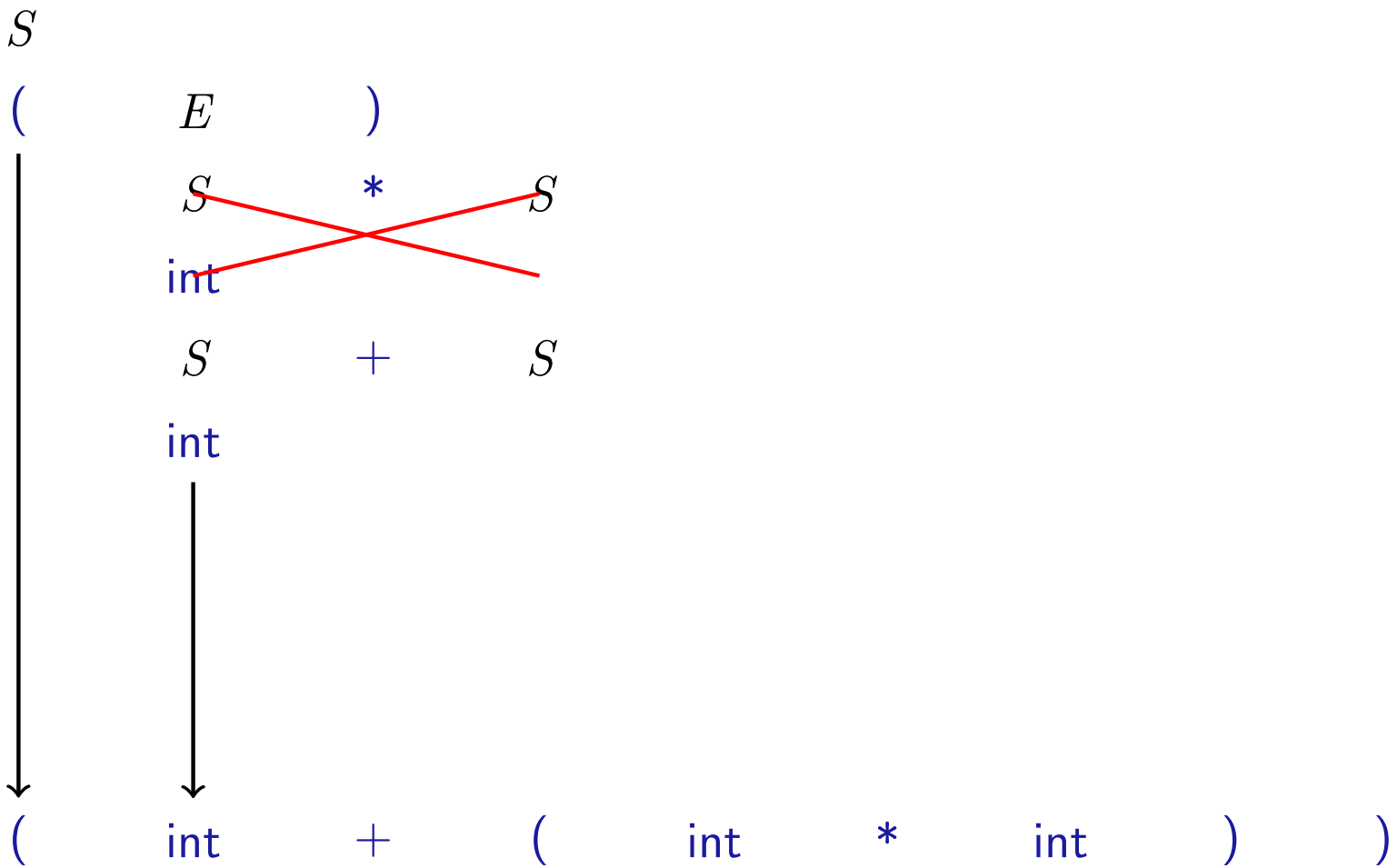
Recursive Descent Parsing

Idea: Recursively compute all expansions of a nonterminal at some input position



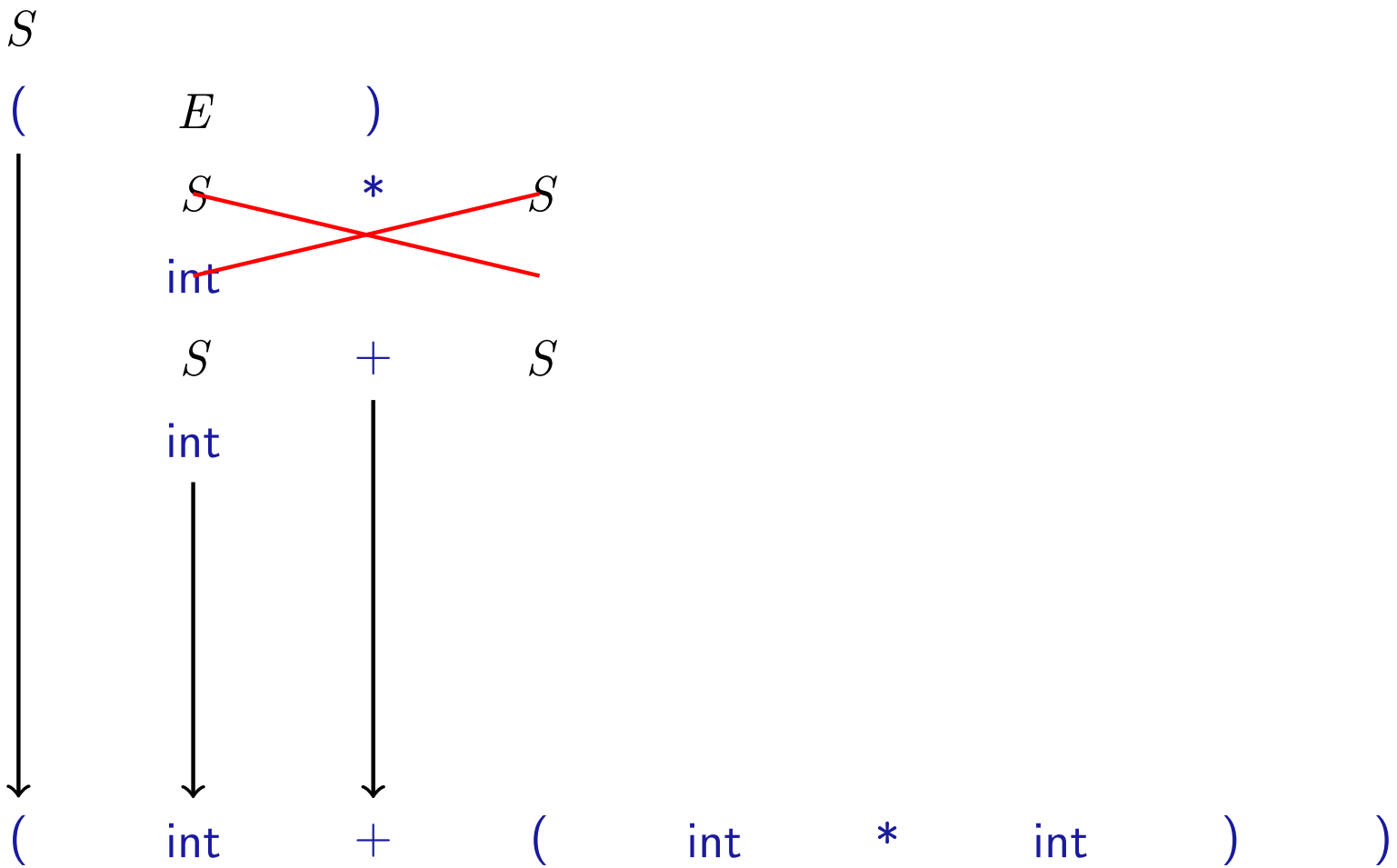
Recursive Descent Parsing

Idea: Recursively compute all expansions of a nonterminal at some input position



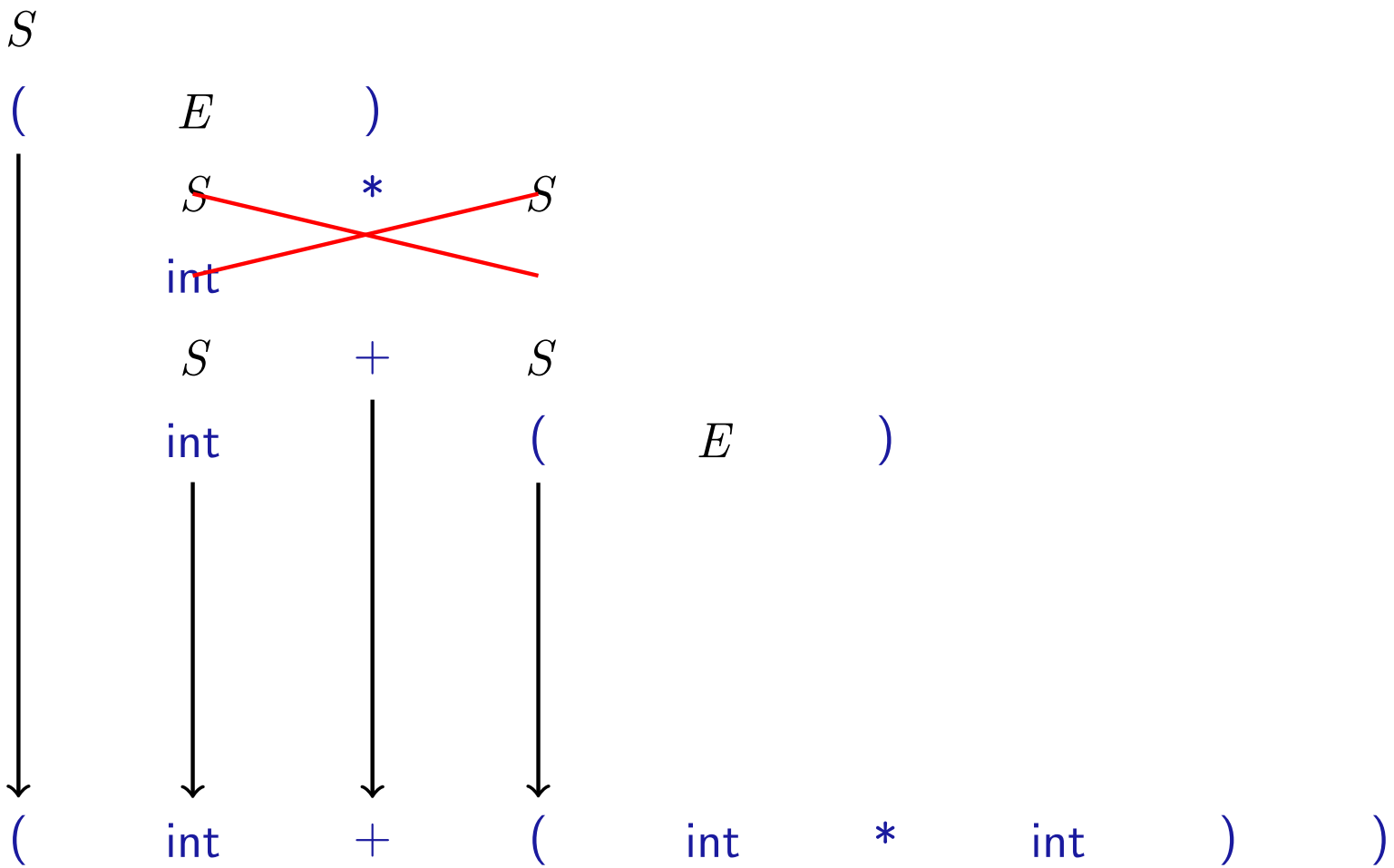
Recursive Descent Parsing

Idea: Recursively compute all expansions of a nonterminal at some input position



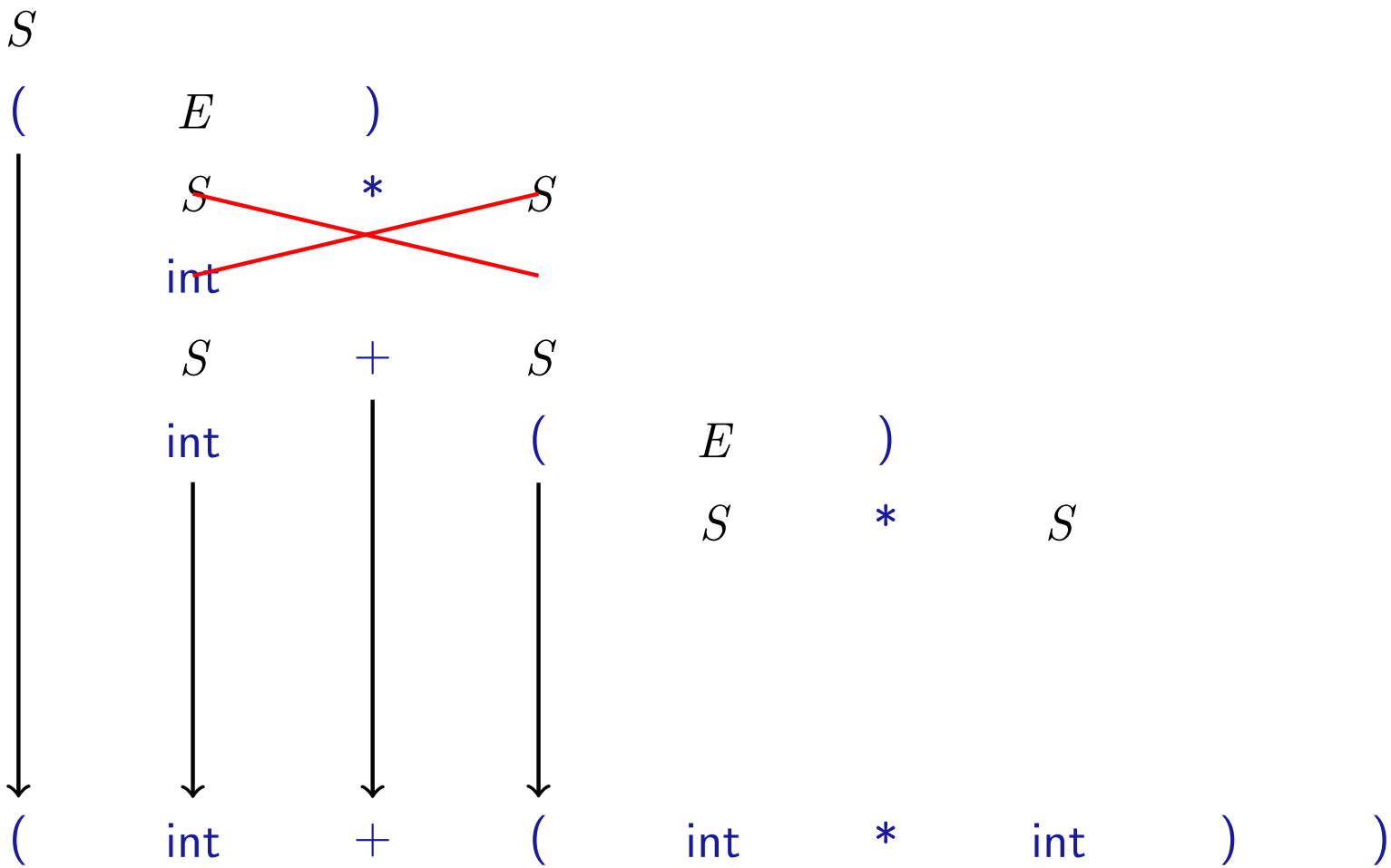
Recursive Descent Parsing

Idea: Recursively compute all expansions of a nonterminal at some input position



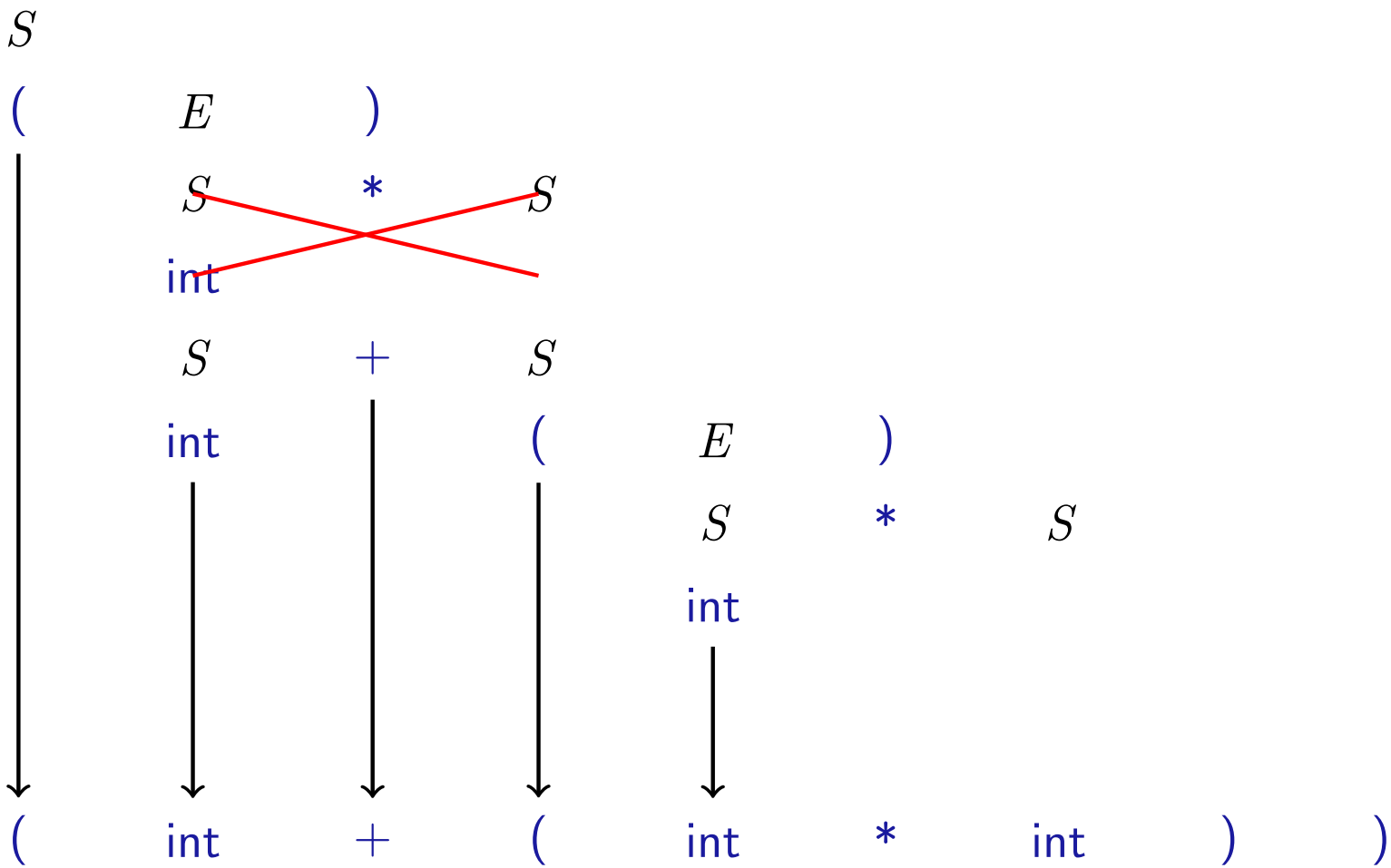
Recursive Descent Parsing

Idea: Recursively compute all expansions of a nonterminal at some input position



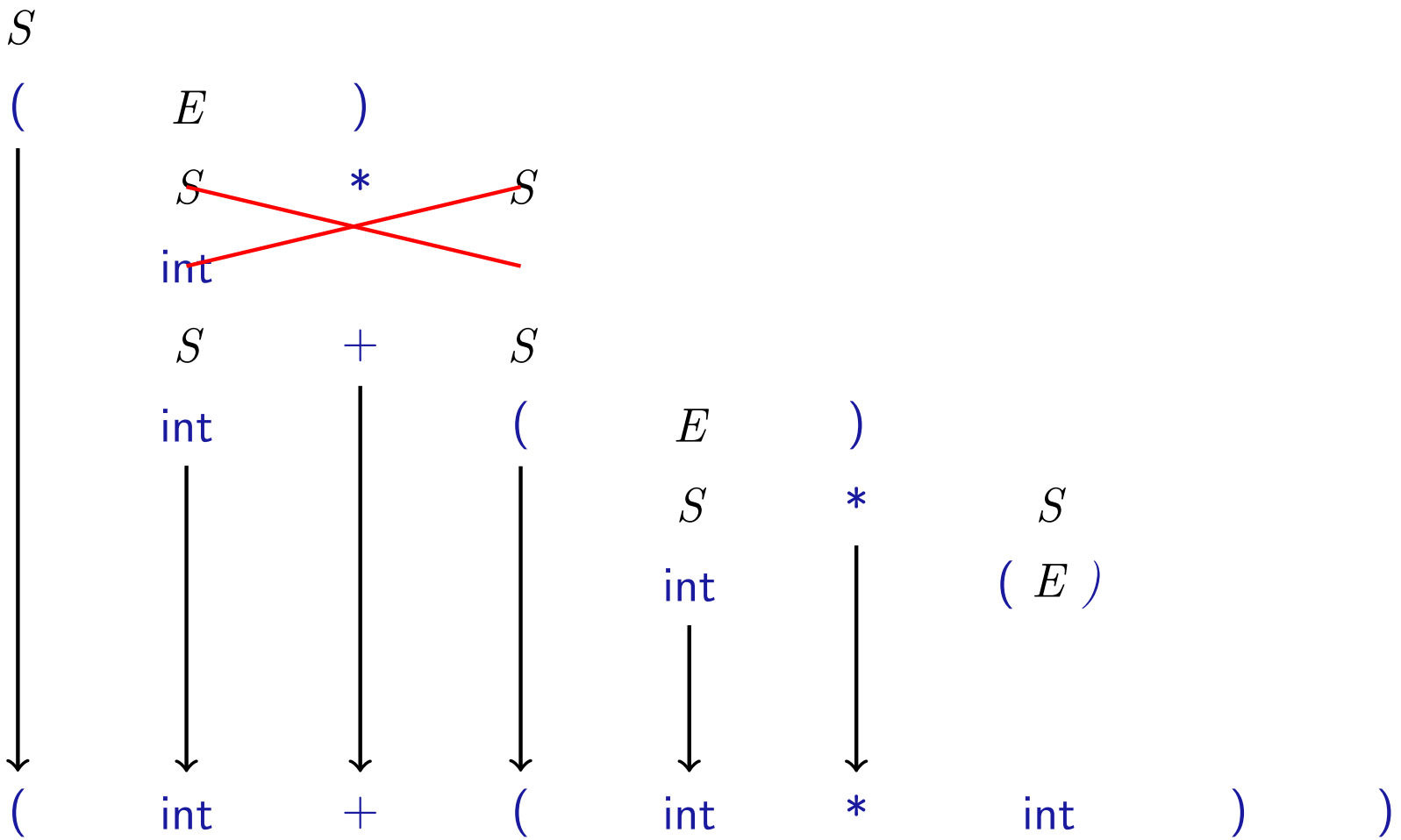
Recursive Descent Parsing

Idea: Recursively compute all expansions of a nonterminal at some input position



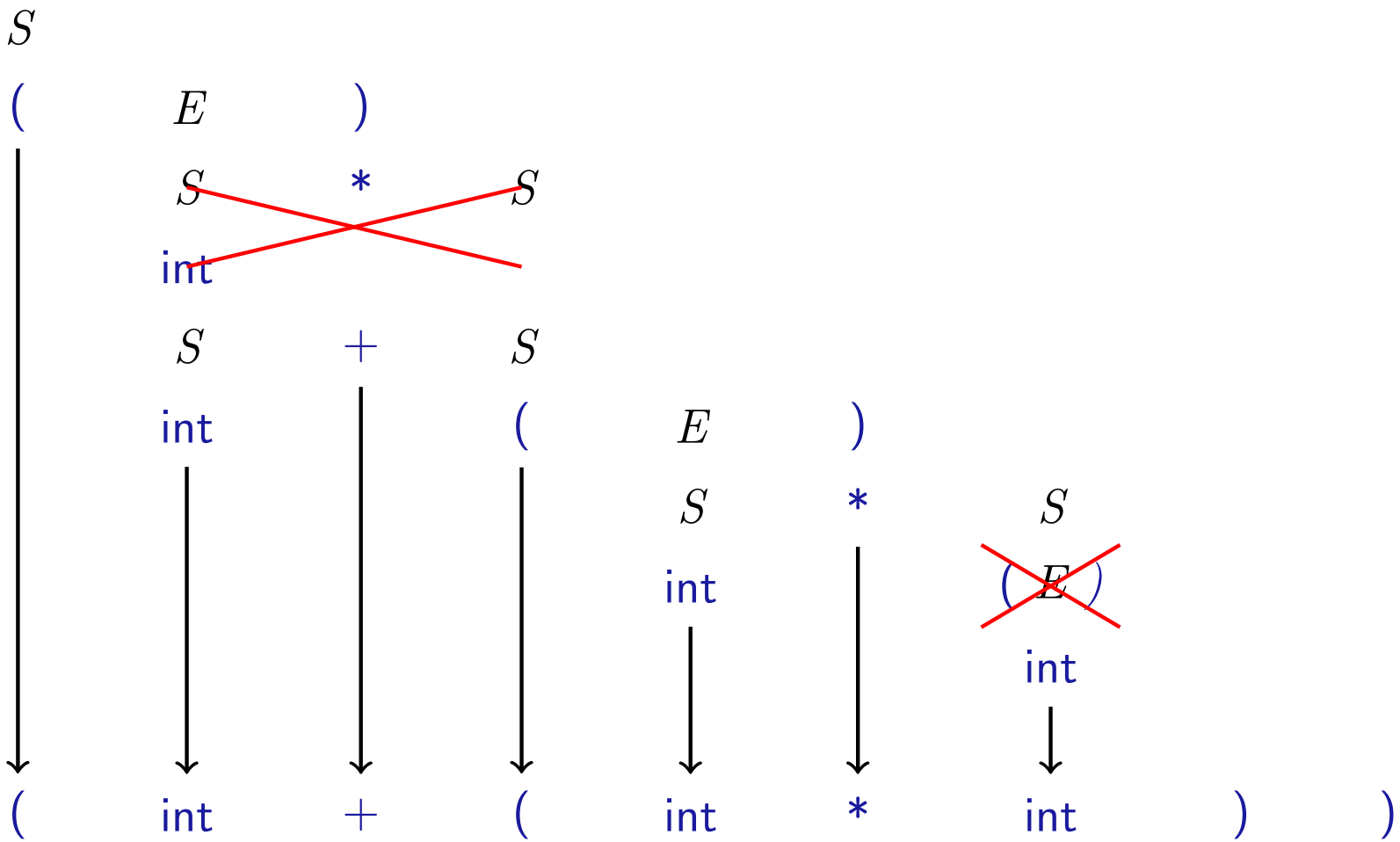
Recursive Descent Parsing

Idea: Recursively compute all expansions of a nonterminal at some input position



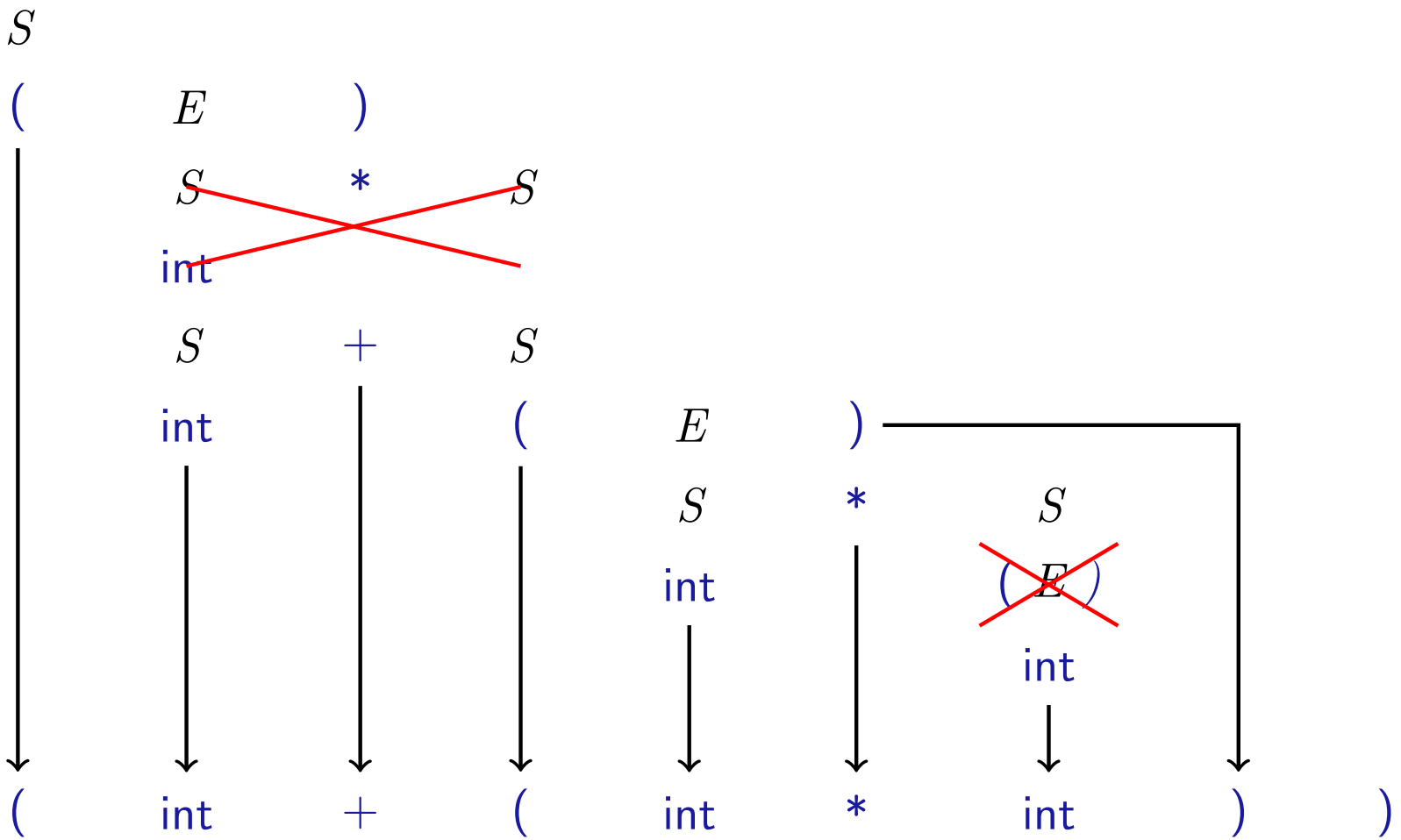
Recursive Descent Parsing

Idea: Recursively compute all expansions of a nonterminal at some input position



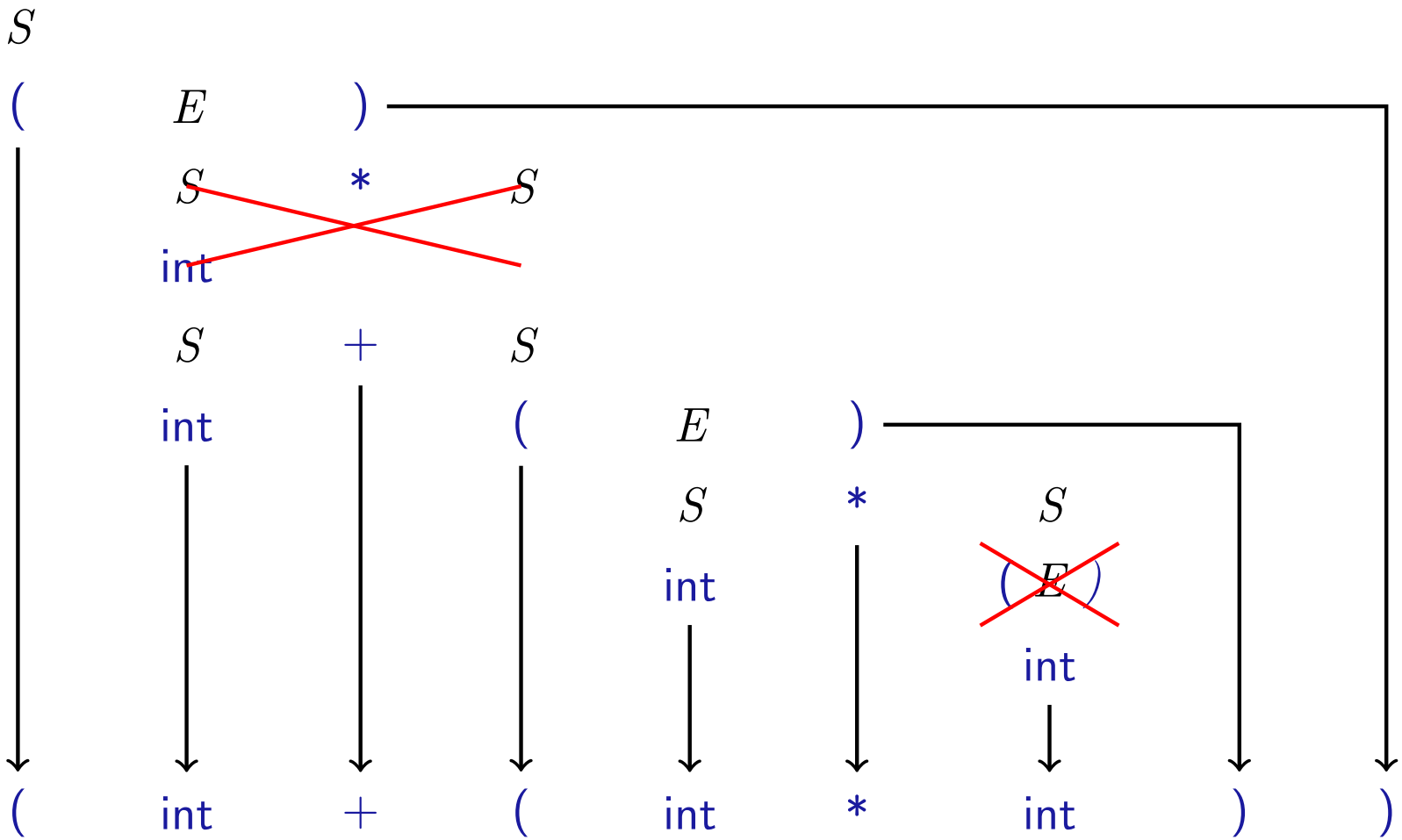
Recursive Descent Parsing

Idea: Recursively compute all expansions of a nonterminal at some input position



Recursive Descent Parsing

Idea: Recursively compute all expansions of a nonterminal at some input position



Recursive Descent Parsing II

- `expand` returns a set of possible end positions
- During expansion of one production
  - Keep track of a set of intermediate positions for the already expanded part
  - Expand the next terminal or nonterminal from every intermediate position
  - If the set of possible intermediate position gets empty, the production fails
- When a production was successfully completed, the intermediate set is added to the set of end positions
- May run into an infinite loop with left-recursive grammars, i.e. grammars where $A \overset{*}{\underset{G}{\Rightarrow}} A\beta$ for some A

Top Down Parsing: Optimizations

- For every nonterminal, compute the set of terminals that can occur in the *first* position
  - Expand only those nonterminals / productions that are compatible with the current input symbol
- Avoid performing duplicate calls with identical nonterminal/position pair: *memoize* previous calls
  - use a data structure whose index are tuples consisting of the function arguments
  - the result of the lookup is the result of a previous call with the same arguments (if available)
  - The memoized method has to be strictly functional for this to work

Epsilon Reduction

- A CFG may contain productions of the form $A \to \epsilon$
- Construct a CFG G' with the same language as G and at most one epsilon production: $S \to \epsilon$

for all nonterminals A with $A \to \epsilon \in P$:
    mark A as ϵ-deriving and add it to the set $\mathcal{Q}$

while $\mathcal{Q}$ is not empty, remove a nonterminal X from $\mathcal{Q}$:
    for all $Y \to \alpha X \beta \in P$, with α or β not empty, add $Y \to \alpha \beta$ to P'
    for all $Y \to X$, if Y is not marked as ϵ-deriving:
        mark Y as ϵ-deriving and add it to $\mathcal{Q}$

if S is ϵ-deriving, add $S \to \epsilon$ to P'
add all non-ϵ productions of P to P'

Chomsky Normal Form

- A context-free grammar is in *Chomsky Normal Form* if:
  - *(i)* it is ϵ-free,
  - *(ii)* all productions have one of two forms:
    - $A \to a$ with $a \in \Sigma$
    - $A \to B\ C$ with $B, C \in N$
- Every CFG can be transformed into a CNF grammar with the same language
- Drawback: The original structure of the parse trees must be reconstructed, if necessary
- The original and transformed grammar are said to be *weakly equivalent*

Chomsky Normal Form II

- Convert an arbitrary CFG into CNF:
  - Introduce new nonterminals and productions $A^a \to a$
  - Replace all occurences of a by A^a
  - Eliminate unary productions $A \to B$:
    add productions where A is replaced by B in the right hand sides
  - Replace productions with more than two symbols on the right hand side by a sequence of productions:
    $A \to R_1\, R_2\, \ldots R_n \Rightarrow$
    $A \to R_1 A^{(1)}, \quad A^{(1)} \to R_2 A^{(2)}, \quad \ldots \quad A^{(n-2)} \to R_{n-1} R_n$

Chart Parsing

- First algorithm independently developed by Cocke, Younger and Kasami (late 60s)
- Given a string w of length n, use an $n \times n$ table to store subderivations (hence chart or tabular parsing)
- Works for all kinds of grammars: left/right recursive, ambiguous
- Storing subderivations avoids duplicate computation: an instance of *dynamic programming*
- polynomial space and time bounds, although an exponential number of parse trees may be encoded!

CYK Parsing

- Input: G in Chomsky normal form, input string $w_1, \ldots, w_n$
- Systematically explore all possible sub-derivations bottom-up
- Use an $n \times n$ array $\mathcal{C}$ such that
  - If nonterminal A is stored in $\mathcal{C}(i,k)$: $A \overset{*}{\underset{G}{\Rightarrow}} w_{i+1}, \ldots, w_k$
  - Maintain a second table $\mathcal{B}$, such that if $j \in \mathcal{B}(i,k)$, for example $A \rightarrow B\ C \in P, A \in \mathcal{C}(i,k), B \in \mathcal{C}(i,j)$ and $C \in \mathcal{C}(j,k)$
  - $\mathcal{B}$ enables us to extract the parse trees
- Implement $\mathcal{C}$ and $\mathcal{B}$ as three-dimensional boolean arrays of size $n \times n \times |N|$ and $n \times n \times n$, respectively

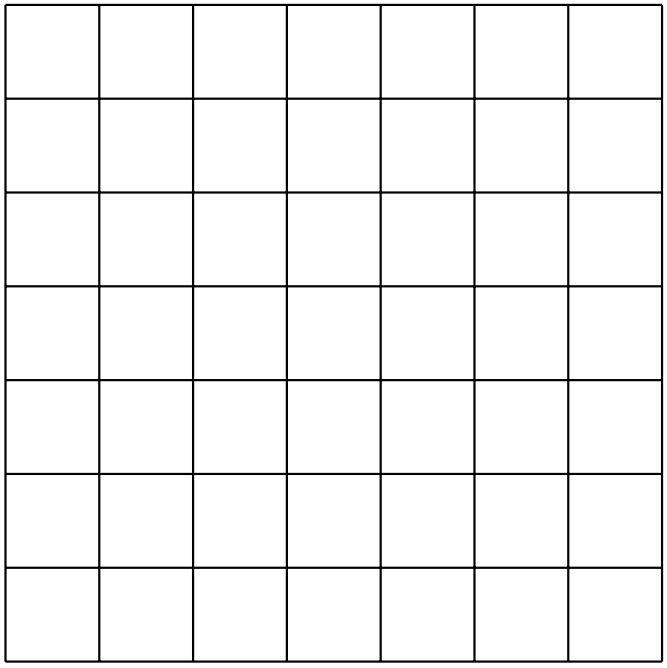
CYK – The Original

for $i = 1$ **to** n **do**
  foreach $R_j \longrightarrow a_i$ **do** $\mathcal{C}(i-1, i, j) = \textbf{true}$

for $len = 2$ **to** n **do** *// Length of new constituent*
  for $start = 0$ **to** $n - l$ **do** *// Start of new constituent*
    for $mid = 1$ **to** $len - 1$ **do** *// Length of first subconstituent*
      foreach $R_a \longrightarrow R_b R_c \in \mathcal{P}$ **do**
        if $\mathcal{C}(start, start + mid, b)$
          $\wedge\ \mathcal{C}(start + mid, start + len, c)$
        then
        $\mathcal{C}(start, start + len, a) = \textbf{true}$
        $\mathcal{B}(start, start + len, start + mid) = \textbf{true}$

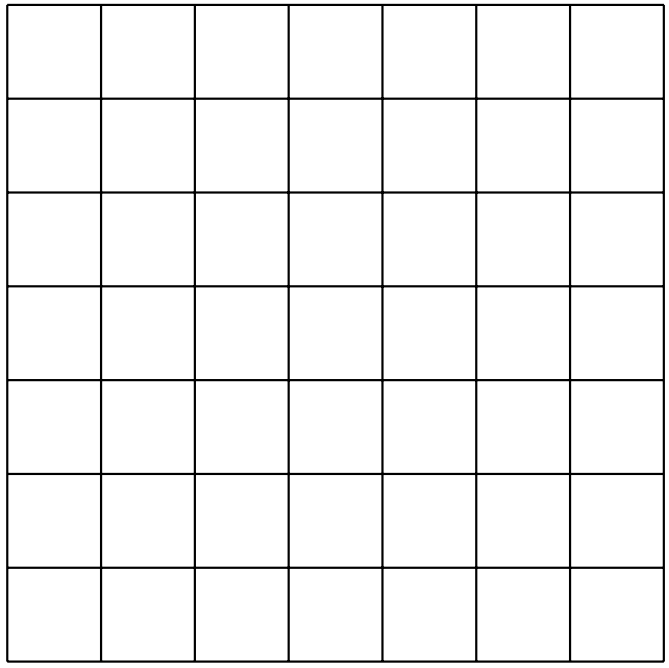
if $\mathcal{C}(1, n, S)$ is **true**, $w \in \mathcal{L}(G)$

CYK Chart Example



$\mathcal{C}$

$\mathcal{B}$



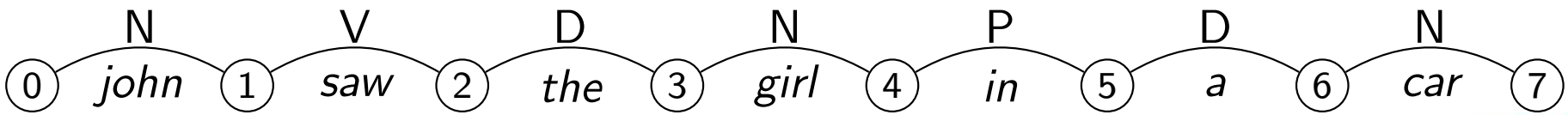
0 *john* 1 *saw* 2 *the* 3 *girl* 4 *in* 5 *a* 6 *car* 7

CYK Chart Example

N						
	V					
		D				
			N			
				P		
					D	
						N

$\mathcal{C}$

$\mathcal{B}$



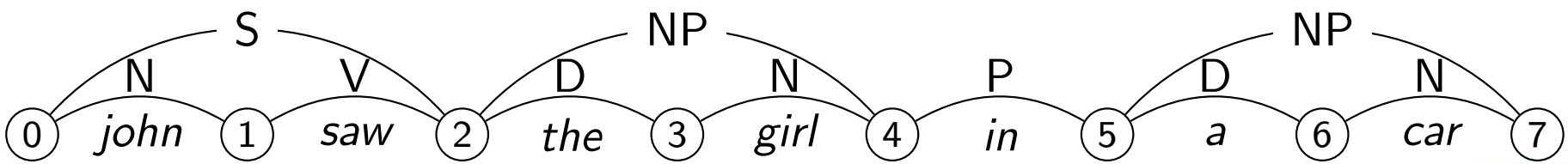
CYK Chart Example

N	S					
	V					
		D	NP			
			N			
				P		
					D	NP
						N

$\mathcal{C}$

$\mathcal{B}$

	1					
			3			
						6



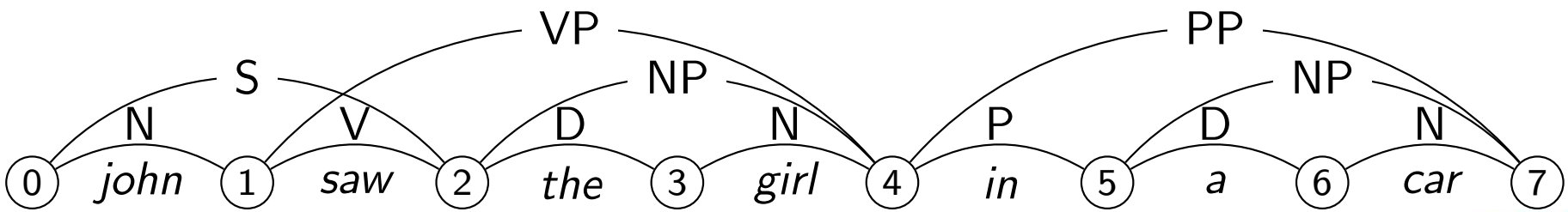
CYK Chart Example

N	S					
	V		VP			
		D	NP			
			N			
				P		PP
					D	NP
						N

$\mathcal{C}$

$\mathcal{B}$

	1					
			2			
			3			
						5
						6



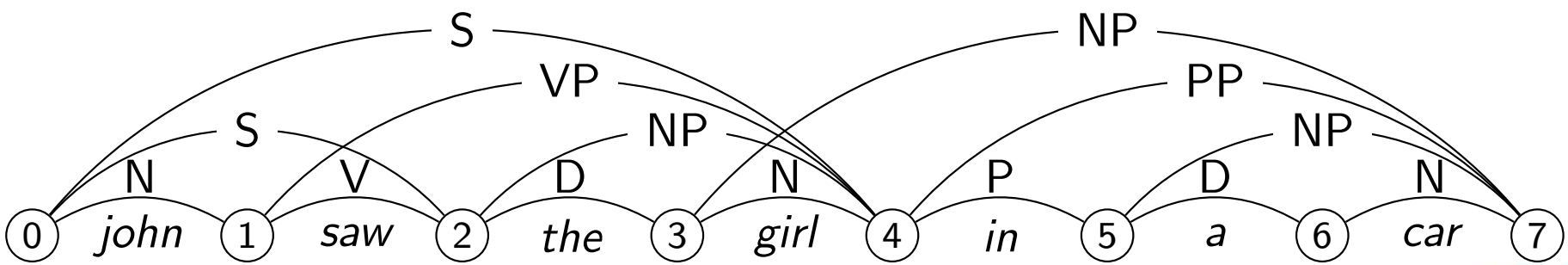
CYK Chart Example

N	S		S			
	V		VP			
		D	NP			
			N			NP
				P		PP
					D	NP
						N

$\mathcal{C}$

$\mathcal{B}$

	1		1			
			2			
			3			
						4
						5
						6



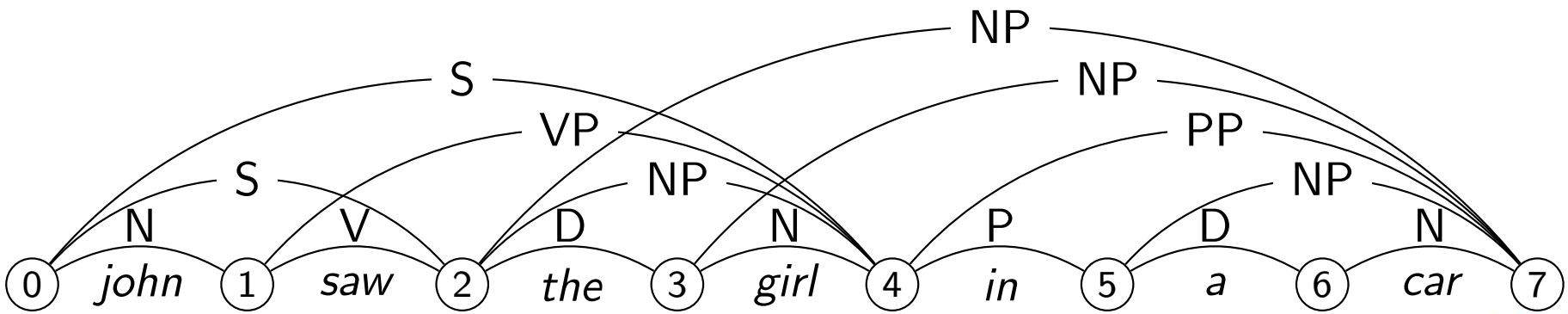
CYK Chart Example

N	S		S			
	V		VP			
		D	NP			NP
			N			NP
				P		PP
					D	NP
						N

$\mathcal{C}$

$\mathcal{B}$

	1		1			
			2			
			3			4
						4
						5
						6



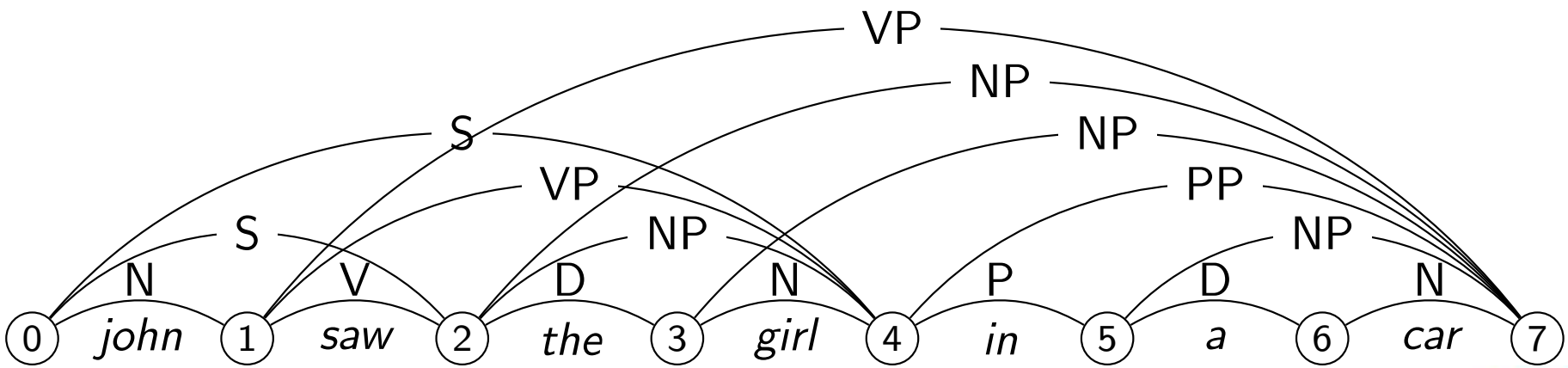
CYK Chart Example

N	S		S			
	V		VP			VP
		D	NP			NP
			N			NP
				P		PP
					D	NP
						N

$\mathcal{C}$

$\mathcal{B}$

	1		1			
			2			2,4
			3			4
						4
						5
						6



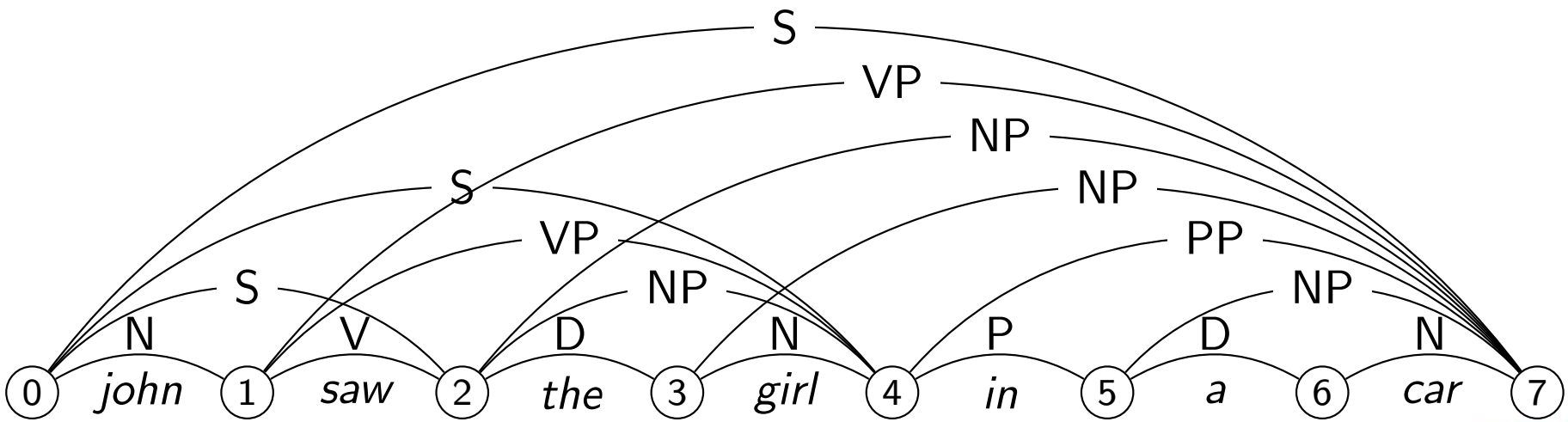
CYK Chart Example

N	S		S			S
	V		VP			VP
		D	NP			NP
			N			NP
				P		PP
					D	NP
						N

$\mathcal{C}$

$\mathcal{B}$

	1		1			1
			2			2,4
			3			4
						4
						5
						6



Example Chart

S $\rightarrow$NP VP|N VP|N V|NP V
VP$\rightarrow$V NP|V N|VP PP
NP$\rightarrow$D N|NP PP|N PP
PP$\rightarrow$P NP|P N
N $\rightarrow$*john, girl, car*
V $\rightarrow$*saw, walks*
P $\rightarrow$*in*
D $\rightarrow$*the, a*

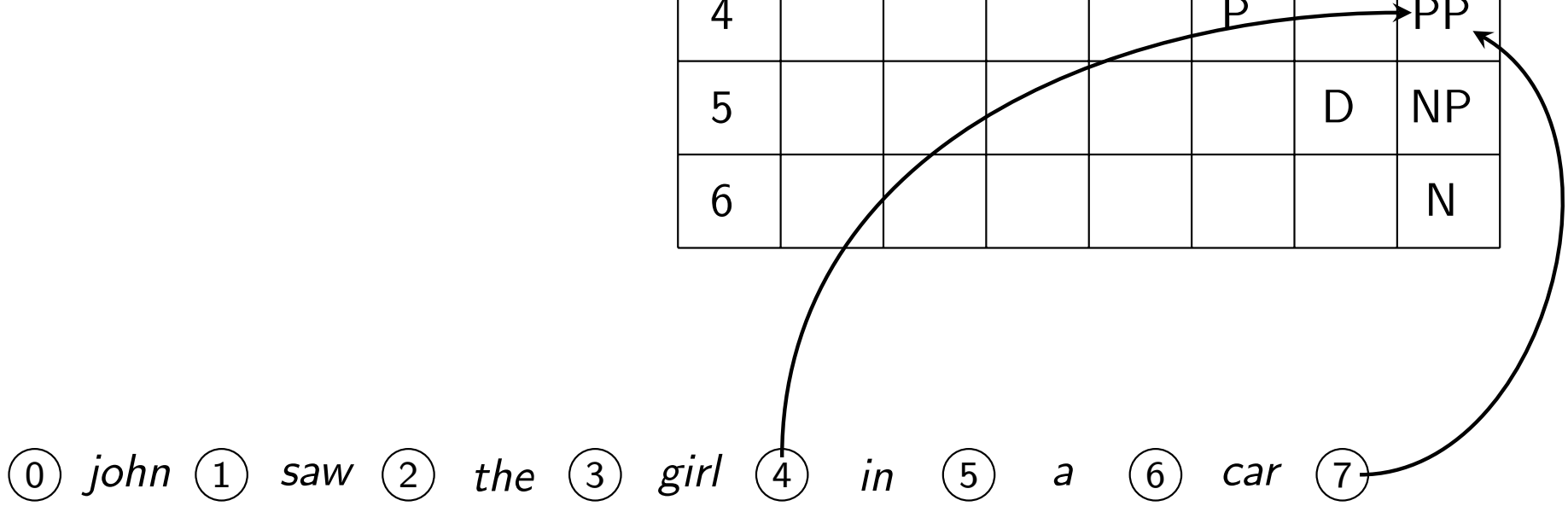
	1	2	3	4	5	6	7
0	N	S		S			S
1		V		VP			VP$_{x2}$
2			D	NP			NP
3				N			NP
4					P		PP
5						D	NP
6							N

(0) *john* (1) *saw* (2) *the* (3) *girl* (4) *in* (5) *a* (6) *car* (7)

Example Chart

S $\rightarrow$NP VP|N VP|N V|NP V
VP$\rightarrow$V NP|V N|VP PP
NP$\rightarrow$D N|NP PP|N PP
PP$\rightarrow$P NP|P N
N $\rightarrow$*john, girl, car*
V $\rightarrow$*saw, walks*
P $\rightarrow$*in*
D $\rightarrow$*the, a*

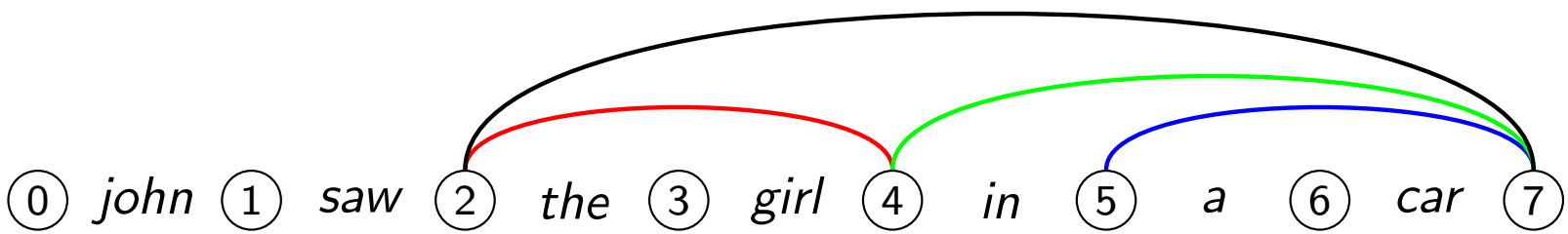
	1	2	3	4	5	6	7
0	N	S		S			S
1		V		VP			VP$_{x2}$
2			D	NP			NP
3				N			NP
4					P		PP
5						D	NP
6							N



Example Chart

S →NP VP|N VP|N V|NP V
VP→V NP|V N|VP PP
NP→D N|NP PP|N PP
PP→P NP|P N
N →*john, girl, car*
V →*saw, walks*
P →*in*
D →*the, a*

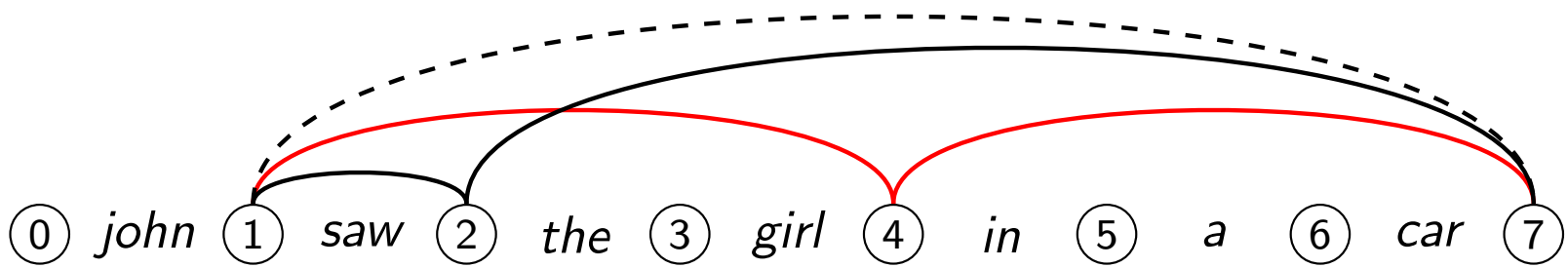
	1	2	3	4	5	6	7
0	N	S		S			S
1		V		VP			VP×2
2			D	NP			NP
3				N			NP
4					P		PP
5						D	NP
6							N



Example Chart

S →NP VP|N VP|N V|NP V
VP→V NP|V N|VP PP
NP→D N|NP PP|N PP
PP→P NP|P N
N →*john, girl, car*
V →*saw, walks*
P →*in*
D →*the, a*

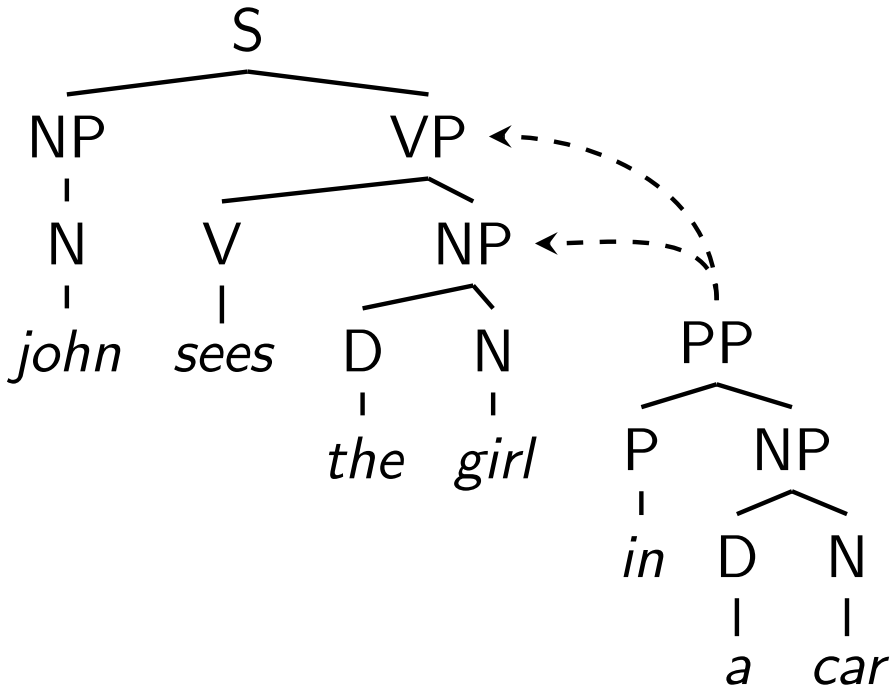
	1	2	3	4	5	6	7
0	N	S		S			S
1		V		VP			VP×2
2			D	NP			NP
3				N			NP
4					P		PP
5						D	NP
6							N



Encoding Ambiguities

$\Sigma = \{john, girl, car, sees, in, the, a\}$ $N = \{S, NP, VP, PP, D, N, V, P\}$

$$P = \left\{ \begin{array}{ll} S \rightarrow NP\ VP, & N \rightarrow john, girl, car \\ VP \rightarrow V | V\ NP | V\ NP\ PP & V \rightarrow sees \\ NP \rightarrow N | D\ N | N\ PP | D\ N\ PP & P \rightarrow in \\ PP \rightarrow P\ NP & D \rightarrow the, a \end{array} \right\}$$



Describing Chart Parsing Algorithms

Parsing algorithm should use *original* grammars for comparability

Describing Chart Parsing Algorithms

Parsing algorithm should use *original* grammars for comparability

- They are described using chart items, which consist of
  - a symbol, derived from the grammar
  - a start and end position from $0, \ldots, n$

Describing Chart Parsing Algorithms

Parsing algorithm should use *original* grammars for comparability

- They are described using chart items, which consist of
  - a symbol, derived from the grammar
  - a start and end position from $0, \ldots, n$
- The symbol of *complete* items is one of $\Sigma \cup N$

Describing Chart Parsing Algorithms

Parsing algorithm should use *original* grammars for comparability

- They are described using chart items, which consist of
  - a symbol, derived from the grammar
  - a start and end position from $0, \ldots, n$
- The symbol of *complete* items is one of $\Sigma \cup N$
- *Incomplete* chart items encode partially filled rules
  - the symbol is a pair (r, i) of rule and *dot position* if $P \ni r : A \to \alpha\beta$ with $|\alpha| = i$
  - write alternatively: $A \to \alpha \bullet \beta$

Bottom-Up Chart Parsing

- How, when and which chart items are created or combined characterizes a parsing algorithm or parsing strategy
- First: A modified variant of Cocke-Younger-Kasami (CYK) algorithm
- Prerequisites: CFG G, input string $w = a_1, \ldots, a_n$
- Data Structures:
  - A $n+1 \times n+1$ chart $\mathcal{C}$, where each cell contains a set of (complete or incomplete) chart items
  - A set of chart items $\mathcal{A}$ (those must still be treated in some way)
- Initialization: add all $(a_i, i-1, i)$ to $\mathcal{A}$ and $\mathcal{C}_{i-1,i}$

Bottom-Up Parsing Algorithm

while $\mathcal{A}$ not empty **do**
  take any (X, i, j) from $\mathcal{A}$ and remove it
  if $X \in \Sigma \cup N$ **then**
    for $A \to X\alpha \in P$ **do**
      $check\_and\_add(A \to X \bullet \alpha, i, j)$
    for $k \in 0, \ldots, i-1$ **do**
      foreach $(A \to \beta \bullet X\alpha, k, i) \in \mathcal{C}$ **do**
        $check\_and\_add(A \to \beta X \bullet \alpha, k, j)$
  else // *incomplete item:* $X \equiv A \to \beta \bullet Y\alpha$
    for $k \in j+1, \ldots, n$ **do**
      if $(Y, j, k) \in \mathcal{C}$ **then** $check\_and\_add(A \to \beta Y \bullet \alpha, i, k)$

proc $check\_and\_add(X \equiv A \to \alpha \bullet \beta, i, j) \equiv$
  if $\beta = \epsilon$ **then** // *complete item*
    if $(A, i, j) \notin \mathcal{C}$ **then** add (A, i, j) to $\mathcal{A}$ and $\mathcal{C}$
  else // *incomplete item*
    if $(A \to \alpha \bullet \beta, i, j) \notin \mathcal{C}$ **then** add $(A \to \alpha \bullet \beta, i, j)$ to $\mathcal{A}$ and $\mathcal{C}$

Bottom-Up: Implementation Details

- How to implement $\mathcal{A}$ and $\mathcal{C}$ efficiently?
- Implementation of the $(n+1)^2$ sets in $\mathcal{C}$:

Bottom-Up: Implementation Details

- How to implement $\mathcal{A}$ and $\mathcal{C}$ efficiently?
- Implementation of the $(n+1)^2$ sets in $\mathcal{C}$:
  - Operations: add single element, contains element

Bottom-Up: Implementation Details

- How to implement $\mathcal{A}$ and $\mathcal{C}$ efficiently?
- Implementation of the $(n+1)^2$ sets in $\mathcal{C}$:
  - Operations: add single element, contains element
  - bit vector of size $|G| := |\Sigma| + |N| + \sum_{P \ni r: A \to \alpha} |A\alpha|$

Bottom-Up: Implementation Details

- How to implement $\mathcal{A}$ and $\mathcal{C}$ efficiently?
- Implementation of the $(n+1)^2$ sets in $\mathcal{C}$:
  - Operations: add single element, contains element
  - bit vector of size $|G| := |\Sigma| + |N| + \sum_{P \ni r : A \to \alpha} |A\alpha|$
- Implementation of the set $\mathcal{A}$:

Bottom-Up: Implementation Details

- How to implement $\mathcal{A}$ and $\mathcal{C}$ efficiently?
- Implementation of the $(n+1)^2$ sets in $\mathcal{C}$:
  - Operations: add single element, contains element
  - bit vector of size $|G| := |\Sigma| + |N| + \sum_{P \ni r: A \to \alpha} |A\alpha|$
- Implementation of the set $\mathcal{A}$:
  - Operations: add , get and remove some element

Bottom-Up: Implementation Details

- How to implement $\mathcal{A}$ and $\mathcal{C}$ efficiently?
- Implementation of the $(n+1)^2$ sets in $\mathcal{C}$:
  - Operations: add single element, contains element
  - bit vector of size $|G| := |\Sigma| + |N| + \sum_{P \ni r:A \to \alpha} |A\alpha|$
- Implementation of the set $\mathcal{A}$:
  - Operations: add , get and remove some element
  - (priority) queue, stack
  - $\mathcal{A}$ is called *agenda* and can be used to implement search strategies
- Keep terminal items separate from the chart for space and time efficiency

Bottom-Up Parsing Algorithm

proc *check_and_add*$(X \equiv A \rightarrow \alpha \bullet \beta, i, j)$
  if $\beta = \epsilon$ **then** *// complete item*
    if $(A, i, j) \notin \mathcal{C}$ **then** add (A, i, j) to $\mathcal{A}$ and $\mathcal{C}$
  else *// incomplete item*
    if $(A \rightarrow \alpha \bullet \beta, i, j) \notin \mathcal{C}$ **then** add $(A \rightarrow \alpha \bullet \beta, i, j)$ to $\mathcal{A}$ and $\mathcal{C}$

Bottom-Up Parsing Algorithm

proc *check_and_add*$(X \equiv A \to \alpha \bullet \beta, i, j)$ all operations $\mathcal{O}(1)$

  if $\beta = \epsilon$ **then** *// complete item*

    if $(A, i, j) \notin \mathcal{C}$ **then** add (A, i, j) to $\mathcal{A}$ and $\mathcal{C}$

  else *// incomplete item*

    if $(A \to \alpha \bullet \beta, i, j) \notin \mathcal{C}$ **then** add $(A \to \alpha \bullet \beta, i, j)$ to $\mathcal{A}$ and $\mathcal{C}$

Bottom-Up Parsing Algorithm

proc *check_and_add*$(X \equiv A \to \alpha \bullet \beta, i, j)$ all operations $\mathcal{O}(1)$
  if $\beta = \epsilon$ **then** *// complete item*
    if $(A, i, j) \notin \mathcal{C}$ **then** add (A, i, j) to $\mathcal{A}$ and $\mathcal{C}$
  else *// incomplete item*
    if $(A \to \alpha \bullet \beta, i, j) \notin \mathcal{C}$ **then** add $(A \to \alpha \bullet \beta, i, j)$ to $\mathcal{A}$ and $\mathcal{C}$

while $\mathcal{A}$ not empty **do**
    take any (X, i, j) from $\mathcal{A}$ and remove it
    if $X \in \Sigma \cup N$ **then**
      for $A \to X\alpha \in P$ **do**
        check_and_add$(A \to X \bullet \alpha, i, j)$
      for $k \in 0, \ldots, i-1$ **do**
        foreach $(A \to \beta \bullet X\alpha, k, i) \in \mathcal{C}$ **do**
          check_and_add$(A \to \beta X \bullet \alpha, k, j)$
    else *// incomplete item:* $X \equiv A \to \beta \bullet Y\alpha$
      for $k \in j+1, \ldots, n$ **do**
        if $(Y, j, k) \in \mathcal{C}$ **then** *check_and_add*$(A \to \beta Y \bullet \alpha, i, k)$

Bottom-Up Parsing Algorithm

proc *check_and_add*$(X \equiv A \to \alpha \bullet \beta, i, j)$ all operations $\mathcal{O}(1)$
  if $\beta = \epsilon$ **then** // *complete item*
    if $(A, i, j) \notin \mathcal{C}$ **then** add (A, i, j) to $\mathcal{A}$ and $\mathcal{C}$
  else // *incomplete item*
    if $(A \to \alpha \bullet \beta, i, j) \notin \mathcal{C}$ **then** add $(A \to \alpha \bullet \beta, i, j)$ to $\mathcal{A}$ and $\mathcal{C}$

while $\mathcal{A}$ not empty **do**
    take any (X, i, j) from $\mathcal{A}$ and remove it max $|G| \times (n+1)^2$
    if $X \in \Sigma \cup N$ **then**
      for $A \to X\alpha \in P$ **do**
        check_and_add$(A \to X \bullet \alpha, i, j)$
      for $k \in 0, \ldots, i-1$ **do**
        foreach $(A \to \beta \bullet X\alpha, k, i) \in \mathcal{C}$ **do**
          check_and_add$(A \to \beta X \bullet \alpha, k, j)$
    else // *incomplete item:* $X \equiv A \to \beta \bullet Y\alpha$
      for $k \in j+1, \ldots, n$ **do**
        if $(Y, j, k) \in \mathcal{C}$ **then** *check_and_add*$(A \to \beta Y \bullet \alpha, i, k)$

Bottom-Up Parsing Algorithm

proc *check_and_add*$(X \equiv A \to \alpha \bullet \beta, i, j)$ all operations $\mathcal{O}(1)$
  if $\beta = \epsilon$ **then** *// complete item*
    if $(A, i, j) \notin \mathcal{C}$ **then** add (A, i, j) to $\mathcal{A}$ and $\mathcal{C}$
  else *// incomplete item*
    if $(A \to \alpha \bullet \beta, i, j) \notin \mathcal{C}$ **then** add $(A \to \alpha \bullet \beta, i, j)$ to $\mathcal{A}$ and $\mathcal{C}$

while $\mathcal{A}$ not empty **do**
    take any (X, i, j) from $\mathcal{A}$ and remove it — $\max |G| \times (n+1)^2$
    if $X \in \Sigma \cup N$ **then**
      for $A \to X\alpha \in P$ **do** — $\max |G|$ times
        check_and_add$(A \to X \bullet \alpha, i, j)$
      for $k \in 0, \ldots, i-1$ **do**
        foreach $(A \to \beta \bullet X\alpha, k, i) \in \mathcal{C}$ **do**
          check_and_add$(A \to \beta X \bullet \alpha, k, j)$
    else *// incomplete item:* $X \equiv A \to \beta \bullet Y\alpha$
      for $k \in j+1, \ldots, n$ **do**
        if $(Y, j, k) \in \mathcal{C}$ **then** *check_and_add*$(A \to \beta Y \bullet \alpha, i, k)$

Bottom-Up Parsing Algorithm

proc *check_and_add*$(X \equiv A \to \alpha \bullet \beta, i, j)$ — all operations $\mathcal{O}(1)$

  if $\beta = \epsilon$ **then** // *complete item*

    if $(A, i, j) \notin \mathcal{C}$ **then** add (A, i, j) to $\mathcal{A}$ and $\mathcal{C}$

  else // *incomplete item*

    if $(A \to \alpha \bullet \beta, i, j) \notin \mathcal{C}$ **then** add $(A \to \alpha \bullet \beta, i, j)$ to $\mathcal{A}$ and $\mathcal{C}$

while $\mathcal{A}$ not empty **do**

  take any (X, i, j) from $\mathcal{A}$ and remove it — max $|G| \times (n+1)^2$

  if $X \in \Sigma \cup N$ **then**

    for $A \to X\alpha \in P$ **do** — max $|G|$ times

      check_and_add$(A \to X \bullet \alpha, i, j)$

    for $k \in 0, \ldots, i-1$ **do** — max n times

      foreach $(A \to \beta \bullet X\alpha, k, i) \in \mathcal{C}$ **do**

        check_and_add$(A \to \beta X \bullet \alpha, k, j)$

  else // *incomplete item:* $X \equiv A \to \beta \bullet Y\alpha$

    for $k \in j+1, \ldots, n$ **do**

      if $(Y, j, k) \in \mathcal{C}$ **then** *check_and_add*$(A \to \beta Y \bullet \alpha, i, k)$

Bottom-Up Parsing Algorithm

proc *check_and_add*$(X \equiv A \to \alpha \bullet \beta, i, j)$ — all operations $\mathcal{O}(1)$

  if $\beta = \epsilon$ **then** // *complete item*

    if $(A, i, j) \notin \mathcal{C}$ **then** add (A, i, j) to $\mathcal{A}$ and $\mathcal{C}$

  else // *incomplete item*

    if $(A \to \alpha \bullet \beta, i, j) \notin \mathcal{C}$ **then** add $(A \to \alpha \bullet \beta, i, j)$ to $\mathcal{A}$ and $\mathcal{C}$

while $\mathcal{A}$ not empty **do**

    take any (X, i, j) from $\mathcal{A}$ and remove it — max $|G| \times (n+1)^2$

    if $X \in \Sigma \cup N$ **then**

      for $A \to X\alpha \in P$ **do** — max $|G|$ times

        check_and_add$(A \to X \bullet \alpha, i, j)$

      for $k \in 0, \ldots, i-1$ **do** — max n times

        foreach $(A \to \beta \bullet X\alpha, k, i) \in \mathcal{C}$ **do** — max $|G|$ times

          check_and_add$(A \to \beta X \bullet \alpha, k, j)$

    else // *incomplete item:* $X \equiv A \to \beta \bullet Y\alpha$

      for $k \in j+1, \ldots, n$ **do**

        if $(Y, j, k) \in \mathcal{C}$ **then** *check_and_add*$(A \to \beta Y \bullet \alpha, i, k)$

Bottom-Up Parsing Algorithm

proc *check_and_add*$(X \equiv A \to \alpha \bullet \beta, i, j)$ — all operations $\mathcal{O}(1)$

  if $\beta = \epsilon$ **then** // *complete item*

    if $(A, i, j) \notin \mathcal{C}$ **then** add (A, i, j) to $\mathcal{A}$ and $\mathcal{C}$

  else // *incomplete item*

    if $(A \to \alpha \bullet \beta, i, j) \notin \mathcal{C}$ **then** add $(A \to \alpha \bullet \beta, i, j)$ to $\mathcal{A}$ and $\mathcal{C}$

while $\mathcal{A}$ not empty **do**

  take any (X, i, j) from $\mathcal{A}$ and remove it — max $|G| \times (n+1)^2$

  if $X \in \Sigma \cup N$ **then**

    for $A \to X\alpha \in P$ **do** — max $|G|$ times

      check_and_add$(A \to X \bullet \alpha, i, j)$

    for $k \in 0, \ldots, i-1$ **do** — max n times

      foreach $(A \to \beta \bullet X\alpha, k, i) \in \mathcal{C}$ **do** — max $|G|$ times

        check_and_add$(A \to \beta X \bullet \alpha, k, j)$

  else // *incomplete item:* $X \equiv A \to \beta \bullet Y\alpha$

    for $k \in j+1, \ldots, n$ **do** — max n times

      if $(Y, j, k) \in \mathcal{C}$ **then** *check_and_add*$(A \to \beta Y \bullet \alpha, i, k)$

Bottom Up – Summary

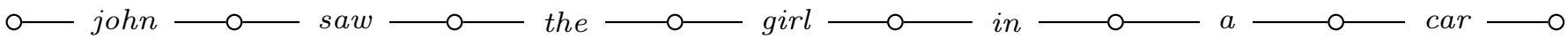
- Polynomial complexity: $\mathcal{O}(|G|^2 n^3)$
- Explores all possible sub-derivations
- Advantageous for *robust parsing*:
  Extract the biggest/best chunks for ungrammatical input
- That a derivation must start at S is not used at all
  - Average time is near or equal to the worst case
  - May lead to poor performance in practice
- Two main steps:
  - if $(X, i, j) \in \mathcal{C}, X \in \Sigma \cup N$ and $A \rightarrow X\alpha \in P$:
    add $(A \rightarrow X \bullet \alpha, i, j)$ to $\mathcal{C}$
  - if $(A \rightarrow \beta \bullet Y\alpha, i, j) \in \mathcal{C}$ and $(Y, j, k) \in \mathcal{C}$:
    add $(A \rightarrow \beta Y \bullet \alpha, i, k)$ to $\mathcal{C}$

Predictive Bottom-Up: Earley Parsing

- Described by J. Earley (1970): Predict Top-Down and Complete Bottom-Up
- Initialize by adding the terminal items *and* $(S \rightarrow \alpha, 0, 0)$ for all $S \rightarrow \bullet\alpha \in P$
- Three main operations:

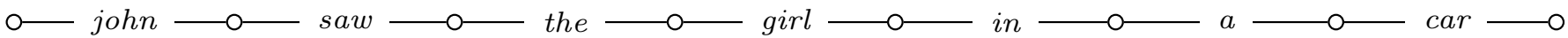
Prediction	if $(A \rightarrow \beta \bullet Y \alpha, i, j) \in \mathcal{C}$, add $(Y \rightarrow \bullet\gamma, j, j)$ to $\mathcal{C}$ for every $Y \rightarrow \gamma \in P$
Scanning	if $(A \rightarrow \beta \bullet a_{j+1} \alpha, i, j) \in \mathcal{C}$, add $(A \rightarrow \beta a_{j+1} \bullet \alpha, i, j+1)$ to $\mathcal{C}$
Completion	if (Y, i, j), $Y \in N$ and $(A \rightarrow \beta \bullet Y \alpha, j, k) \in \mathcal{C}$, add $(A \rightarrow \beta Y \bullet \alpha, i, k)$

Earley Parsing Example



S→ • NP VP

Earley Parsing Example



S → • NP VP

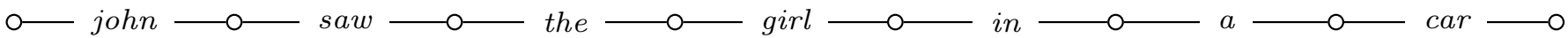
NP → • D N PP

NP → • N

NP → • D N

NP → • N PP

Earley Parsing Example



S → • NP VP

NP → • D N PP

NP → • N

NP → • D N

NP → • N PP

N → • *john*

N → • *girl*

N → • *car*

D → • *the*

D → • *a*

Earley Parsing Example

S → • NP VP

NP → • D N PP

NP → • N

NP → • D N

NP → • N PP

N → • *john*

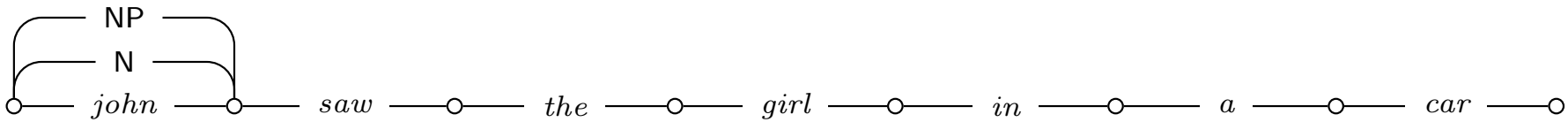
N → • *girl*

N → • *car*

D → • *the*

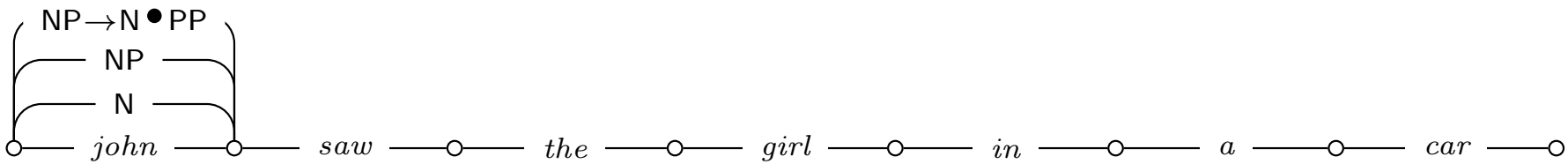
D → • *a*

Earley Parsing Example



S → • NP VP
NP → • D N PP
NP → • N
NP → • D N
NP → • N PP
N → • *john*
N → • *girl*
N → • *car*
D → • *the*
D → • *a*

Earley Parsing Example



S→ • NP VP PP→ • P NP

NP→ • D N PP

NP→ • N

NP→ • D N

NP→ • N PP

N→ • *john*

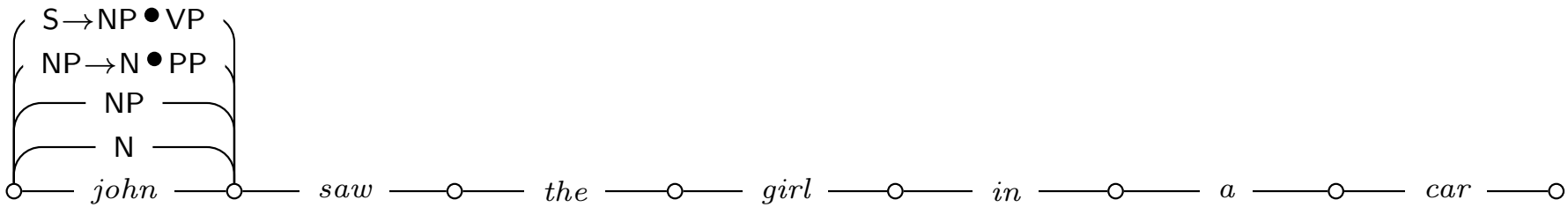
N→ • *girl*

N→ • *car*

D→ • *the*

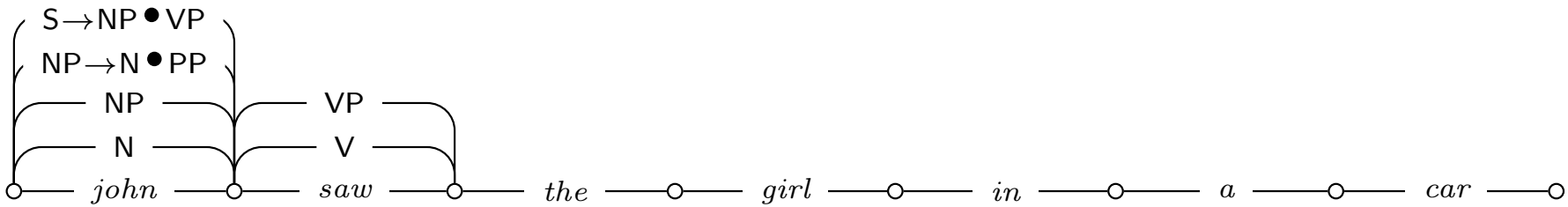
D→ • *a*

Earley Parsing Example



S→ • NP VP
NP→ • D N PP
NP→ • D N
NP→ • N PP
NP→ • N
N→ • *john*
N→ • *girl*
N→ • *car*
D→ • *the*
D→ • *a*
PP→ • P NP
VP→ • V
VP→ • V NP PP
VP→ • V NP
V→ • *saw*

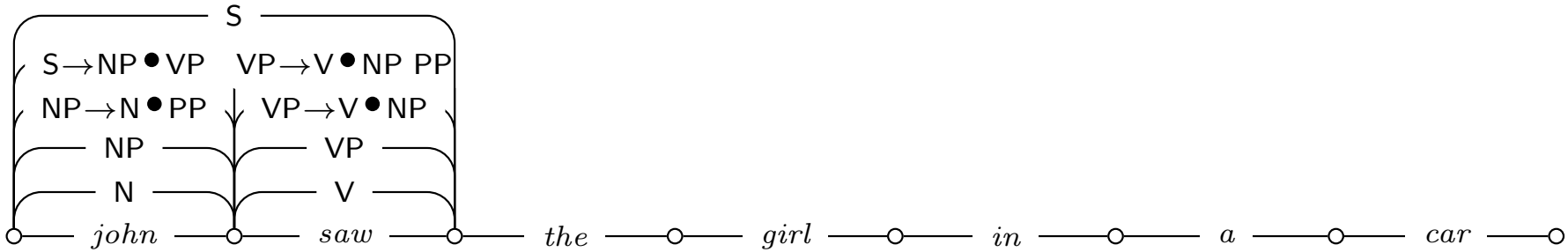
Earley Parsing Example



S→ • NP VP
NP→ • D N PP
NP→ • N
NP→ • D N
NP→ • N PP
N→ • *john*
N→ • *girl*
N→ • *car*
D→ • *the*
D→ • *a*

PP→ • P NP
VP→ • V
VP→ • V NP PP
VP→ • V NP
V→ • *saw*

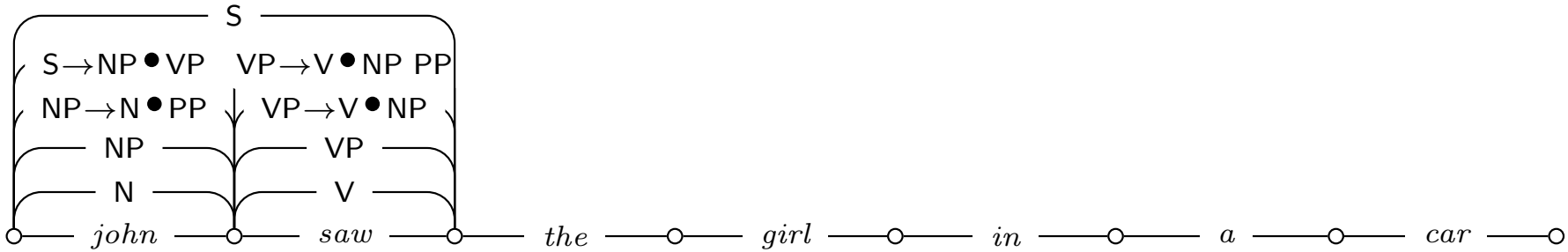
Earley Parsing Example



S→ • NP VP
NP→ • D N PP
NP→ • N
NP→ • D N
NP→ • N PP
N→ • *john*
N→ • *girl*
N→ • *car*
D→ • *the*
D→ • *a*

PP→ • P NP
VP→ • V
VP→ • V NP PP
VP→ • V NP
V→ • *saw*

Earley Parsing Example

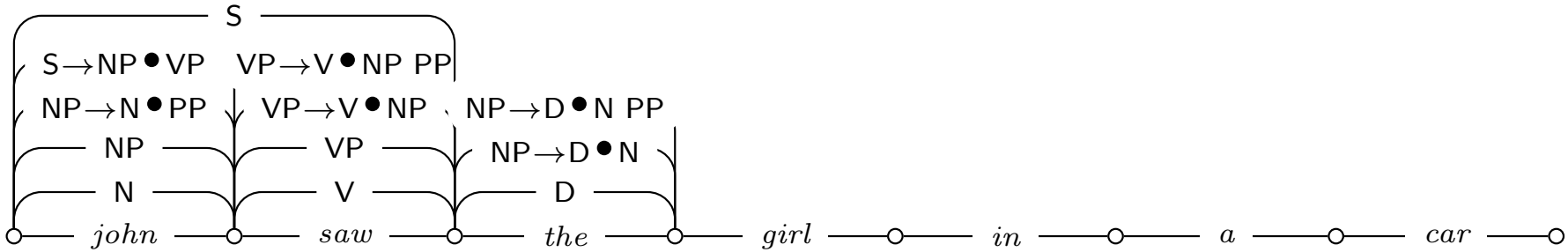


S→ • NP VP
NP→ • D N PP
NP→ • D N
NP→ • N
NP→ • D N
NP→ • N PP
N→ • *john*
N→ • *girl*
N→ • *car*
D→ • *the*
D→ • *a*

PP→ • P NP
VP→ • V
VP→ • V NP PP
VP→ • V NP
V→ • *saw*

NP→ • D N PP
NP→ • D N
NP→ • N
NP→ • N PP
N→ • *john*
N→ • *girl*
N→ • *car*
D→ • *the*
D→ • *a*

Earley Parsing Example

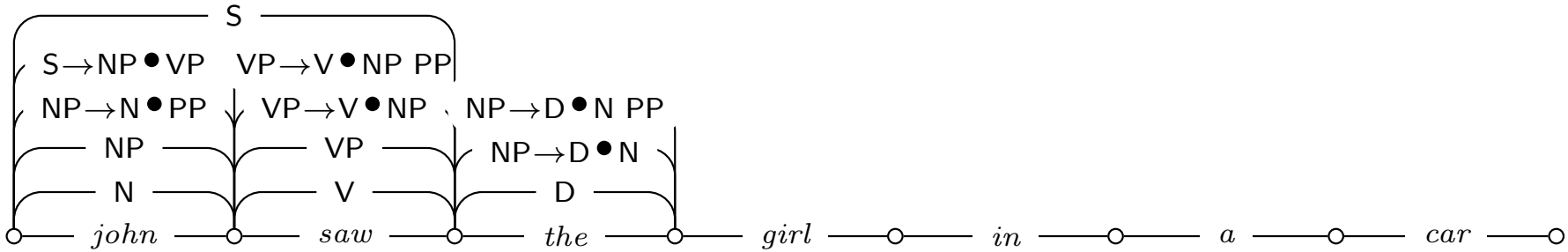


S→ • NP VP
NP→ • D N PP
NP→ • D N
NP→ • N
NP→ • D N
NP→ • N PP
N→ • *john*
N→ • *girl*
N→ • *car*
D→ • *the*
D→ • *a*

PP→ • P NP
VP→ • V
VP→ • V NP PP
VP→ • V NP
V→ • *saw*

NP→ • D N PP
NP→ • D N
NP→ • N
NP→ • N PP
N→ • *john*
N→ • *girl*
N→ • *car*
D→ • *the*
D→ • *a*

Earley Parsing Example



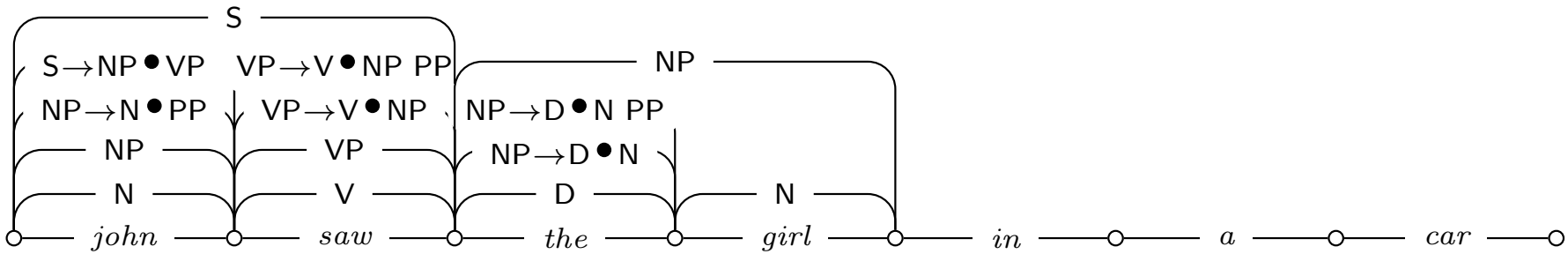
S→ • NP VP
NP→ • D N PP
NP→ • N
NP→ • D N
NP→ • N PP
N→ • *john*
N→ • *girl*
N→ • *car*
D→ • *the*
D→ • *a*

PP→ • P NP
VP→ • V
VP→ • V NP PP
VP→ • V NP
V→ • *saw*

NP→ • D N PP
NP→ • D N
NP→ • N
NP→ • N PP
N→ • *john*
N→ • *girl*
N→ • *car*
D→ • *the*
D→ • *a*

N→ • *john*
N→ • *girl*
N→ • *car*

Earley Parsing Example



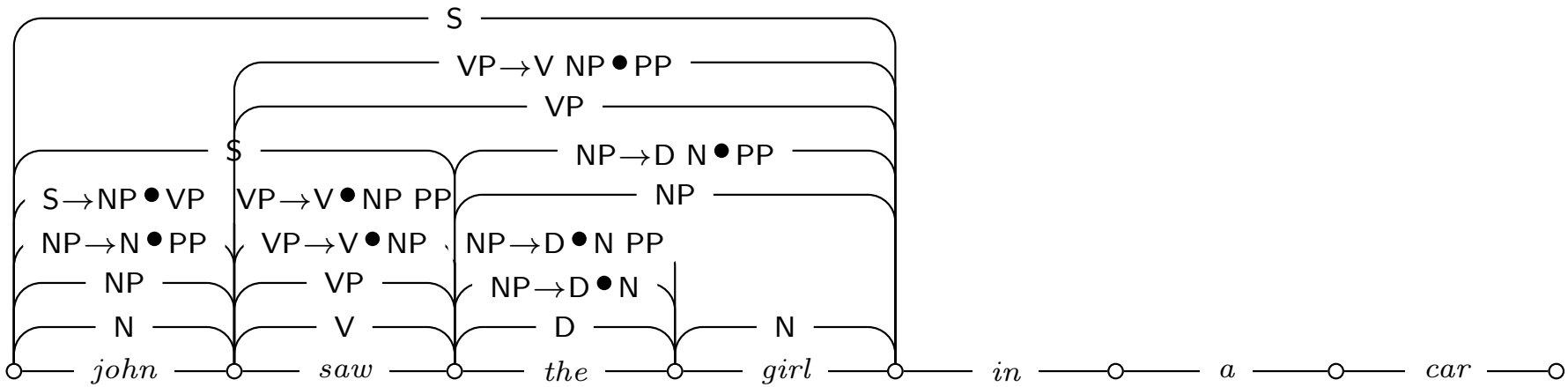
S→ • NP VP
NP→ • D N PP
NP→ • N
NP→ • D N
NP→ • N PP
N→ • *john*
N→ • *girl*
N→ • *car*
D→ • *the*
D→ • *a*

PP→ • P NP
VP→ • V
VP→ • V NP PP
VP→ • V NP
V→ • *saw*

NP→ • D N PP
NP→ • D N
NP→ • N
NP→ • N PP
N→ • *john*
N→ • *girl*
N→ • *car*
D→ • *the*
D→ • *a*

N→ • *john*
N→ • *girl*
N→ • *car*

Earley Parsing Example



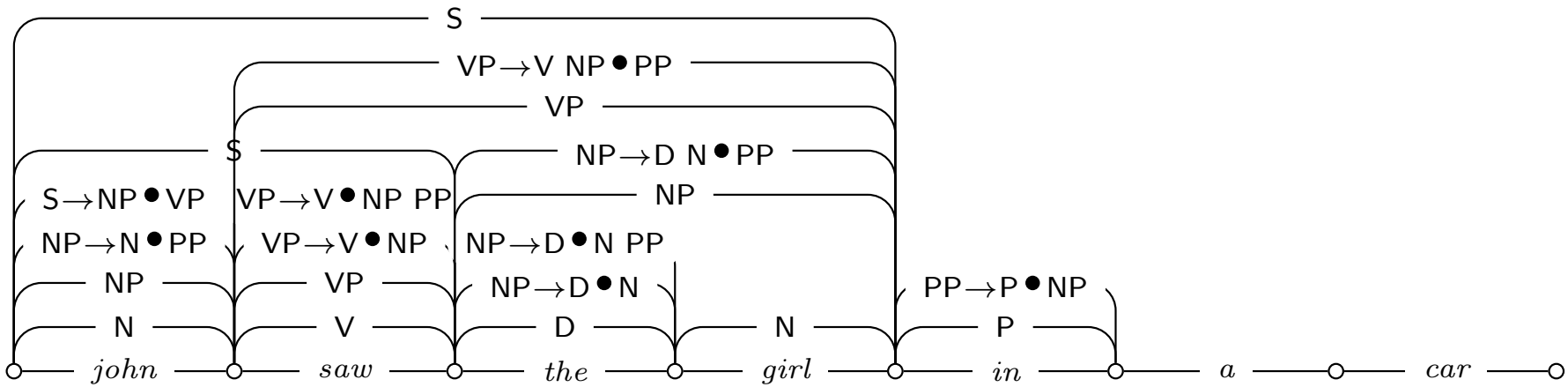
S→ • NP VP
NP→ • D N PP
NP→ • N
NP→ • D N
NP→ • N PP
N→ • *john*
N→ • *girl*
N→ • *car*
D→ • *the*
D→ • *a*

PP→ • P NP
VP→ • V
VP→ • V NP PP
VP→ • V NP
V→ • *saw*

NP→ • D N PP
NP→ • D N
NP→ • N
NP→ • N PP
N→ • *john*
N→ • *girl*
N→ • *car*
D→ • *the*
D→ • *a*

N→ • *john*
N→ • *girl*
N→ • *car*

Earley Parsing Example



S→ • NP VP
NP→ • D N PP
NP→ • N
NP→ • D N
NP→ • N PP
N→ • *john*
N→ • *girl*
N→ • *car*
D→ • *the*
D→ • *a*

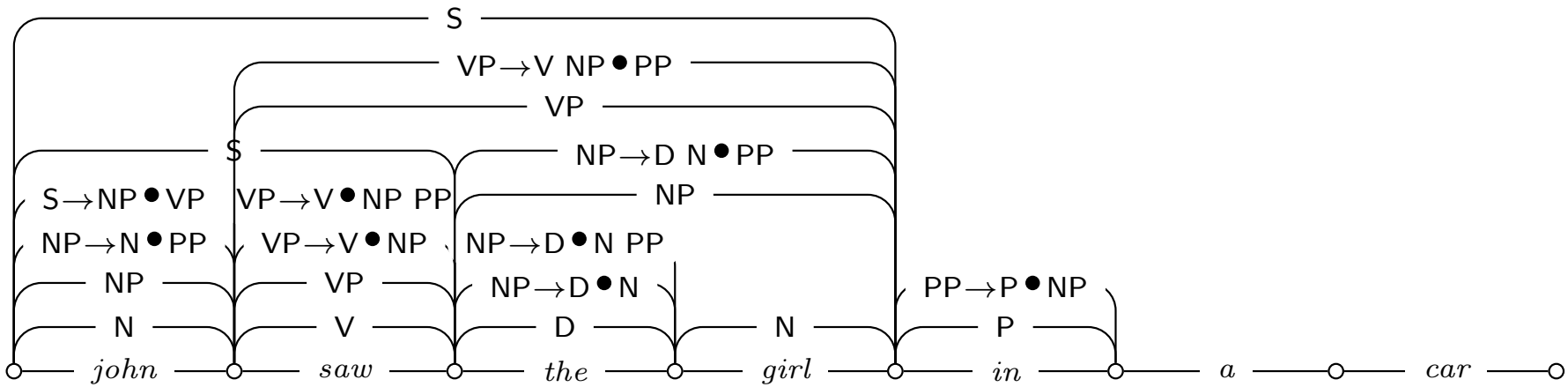
PP→ • P NP
VP→ • V
VP→ • V NP PP
VP→ • V NP
V→ • *saw*

NP→ • D N PP
NP→ • D N
NP→ • N
NP→ • N PP
N→ • *john*
N→ • *girl*
N→ • *car*
D→ • *the*
D→ • *a*

N→ • *john*
N→ • *girl*
N→ • *car*

P→ • *in*
PP→ • P NP

Earley Parsing Example



S→ • NP VP
NP→ • D N PP
NP→ • N
NP→ • D N
NP→ • N PP
N→ • *john*
N→ • *girl*
N→ • *car*
D→ • *the*
D→ • *a*

PP→ • P NP
VP→ • V
VP→ • V NP PP
VP→ • V NP
V→ • *saw*

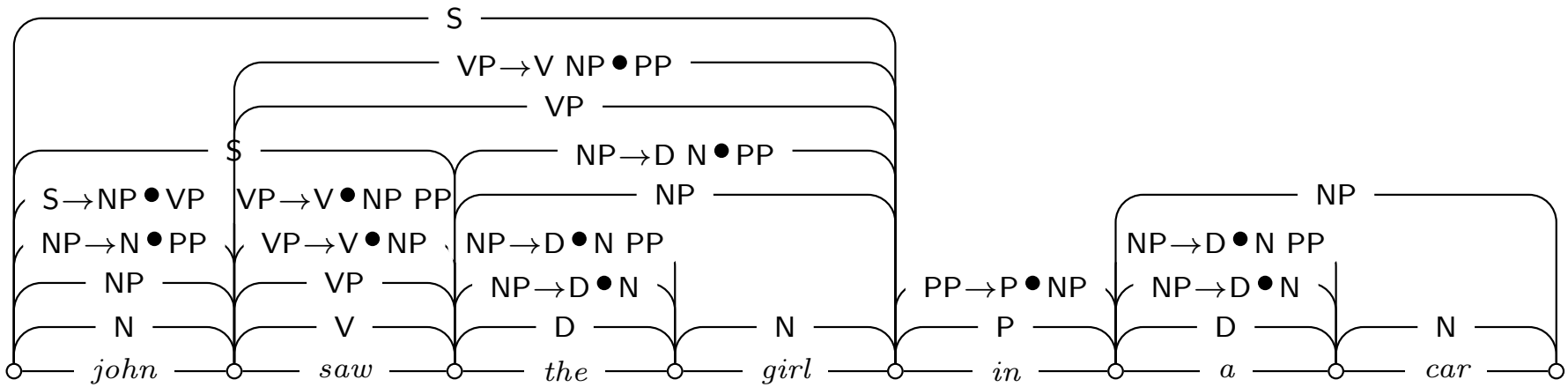
NP→ • D N PP
NP→ • D N
NP→ • N
NP→ • N PP
N→ • *john*
N→ • *girl*
N→ • *car*
D→ • *the*
D→ • *a*

N→ • *john*
N→ • *girl*
N→ • *car*

P→ • *in*
PP→ • P NP

NP→ • D N PP
NP→ • D N
NP→ • N PP
NP→ • N
N→ • *john*
N→ • *girl*
N→ • *car*
D→ • *the*
D→ • *a*

Earley Parsing Example



S→ • NP VP
NP→ • D N PP
NP→ • N
NP→ • D N
NP→ • N PP
N→ • *john*
N→ • *girl*
N→ • *car*
D→ • *the*
D→ • *a*

PP→ • P NP
VP→ • V
VP→ • V NP PP
VP→ • V NP
V→ • *saw*

NP→ • D N PP
NP→ • D N
NP→ • N
NP→ • N PP
N→ • *john*
N→ • *girl*
N→ • *car*
D→ • *the*
D→ • *a*

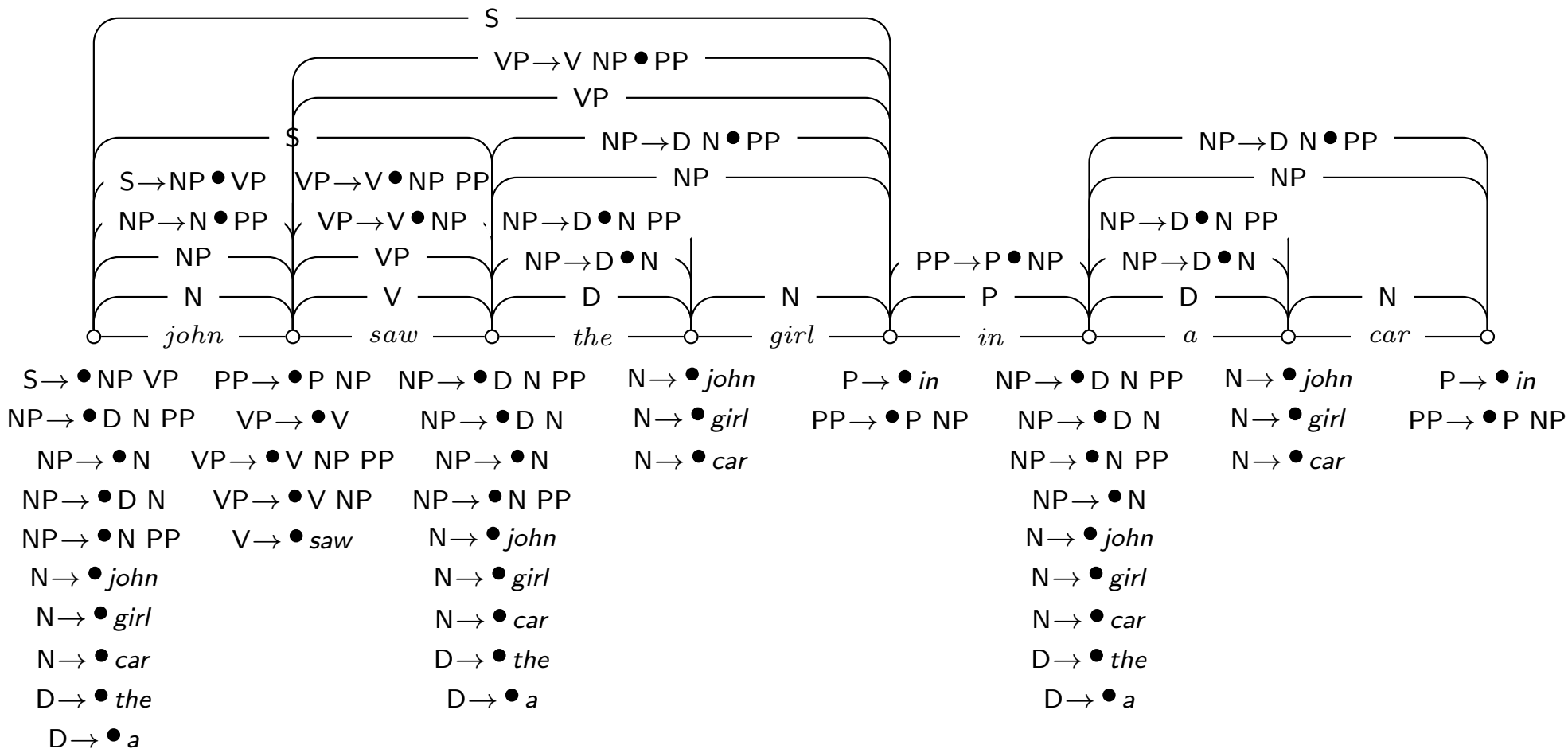
N→ • *john*
N→ • *girl*
N→ • *car*

P→ • *in*
PP→ • P NP

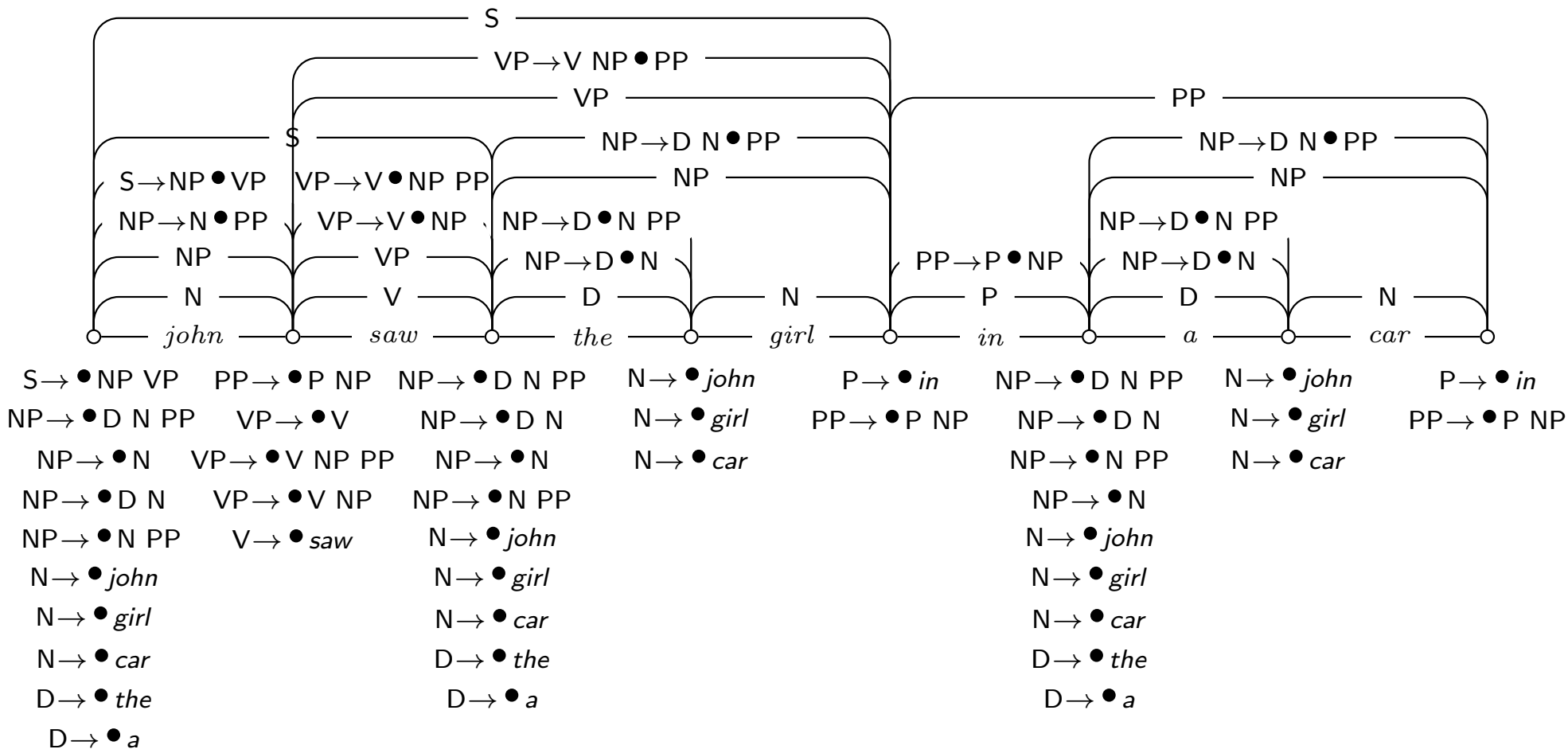
NP→ • D N PP
NP→ • D N
NP→ • N PP
NP→ • N
N→ • *john*
N→ • *girl*
N→ • *car*
D→ • *the*
D→ • *a*

N→ • *john*
N→ • *girl*
N→ • *car*

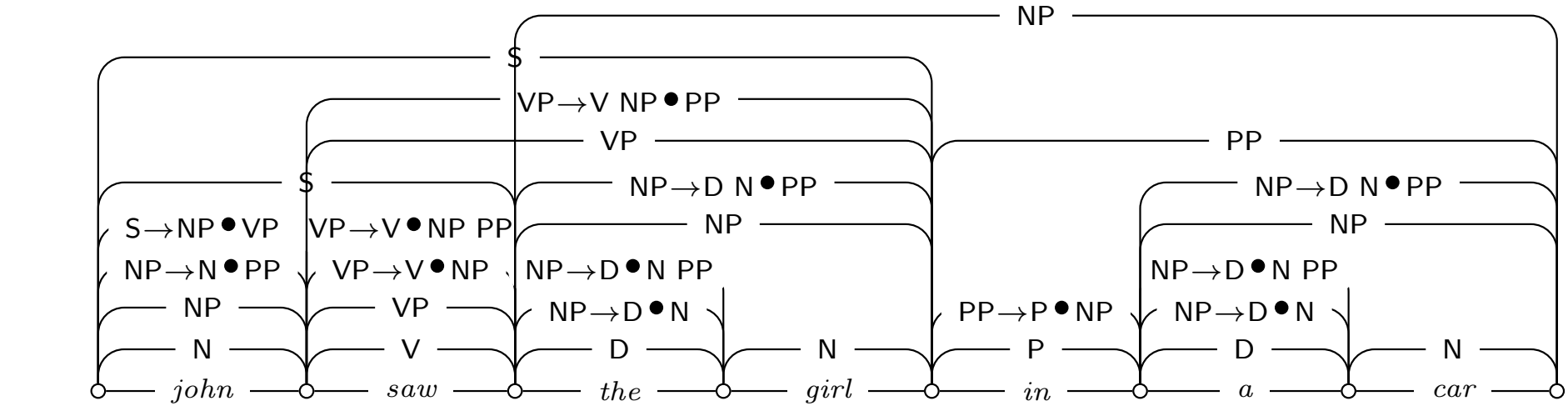
Earley Parsing Example



Earley Parsing Example



Earley Parsing Example



S→ • NP VP
NP→ • D N PP
NP→ • N
NP→ • D N
NP→ • N PP
N→ • *john*
N→ • *girl*
N→ • *car*
D→ • *the*
D→ • *a*

PP→ • P NP
VP→ • V
VP→ • V NP PP
VP→ • V NP
V→ • *saw*

NP→ • D N PP
NP→ • D N
NP→ • N
NP→ • N PP
N→ • *john*
N→ • *girl*
N→ • *car*
D→ • *the*
D→ • *a*

N→ • *john*
N→ • *girl*
N→ • *car*

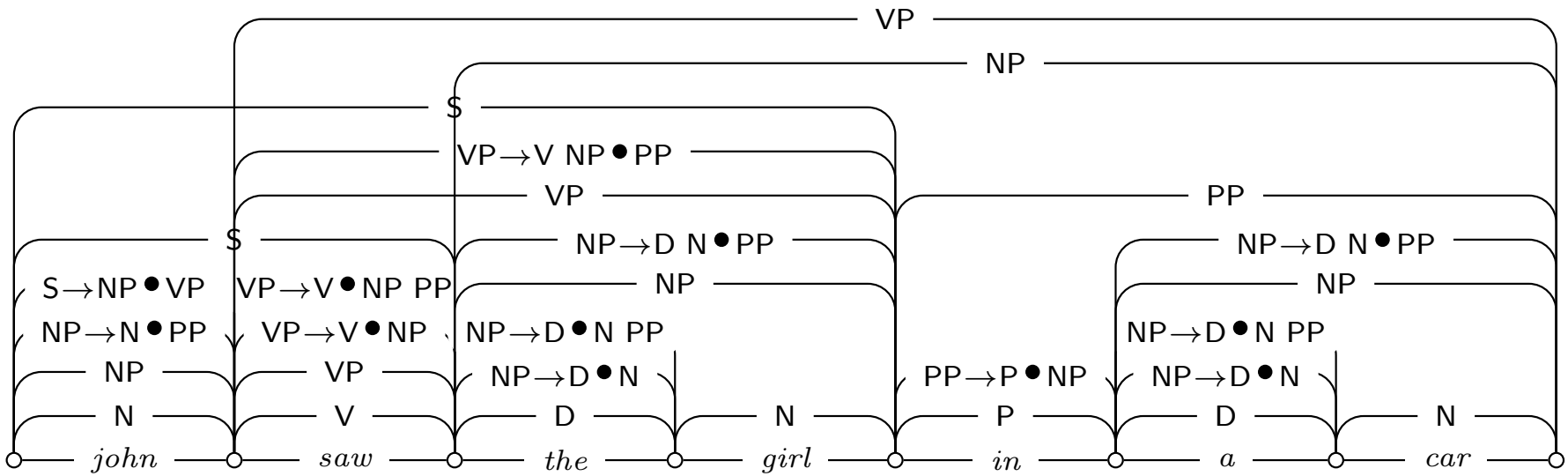
P→ • *in*
PP→ • P NP

NP→ • D N PP
NP→ • D N
NP→ • N PP
NP→ • N
N→ • *john*
N→ • *girl*
N→ • *car*
D→ • *the*
D→ • *a*

N→ • *john*
N→ • *girl*
N→ • *car*

P→ • *in*
PP→ • P NP

Earley Parsing Example



S→ • NP VP
NP→ • D N PP
NP→ • N
NP→ • D N
NP→ • N PP
N→ • *john*
N→ • *girl*
N→ • *car*
D→ • *the*
D→ • *a*

PP→ • P NP
VP→ • V
VP→ • V NP PP
VP→ • V NP
V→ • *saw*

NP→ • D N PP
NP→ • D N
NP→ • N
NP→ • N PP
N→ • *john*
N→ • *girl*
N→ • *car*
D→ • *the*
D→ • *a*

N→ • *john*
N→ • *girl*
N→ • *car*

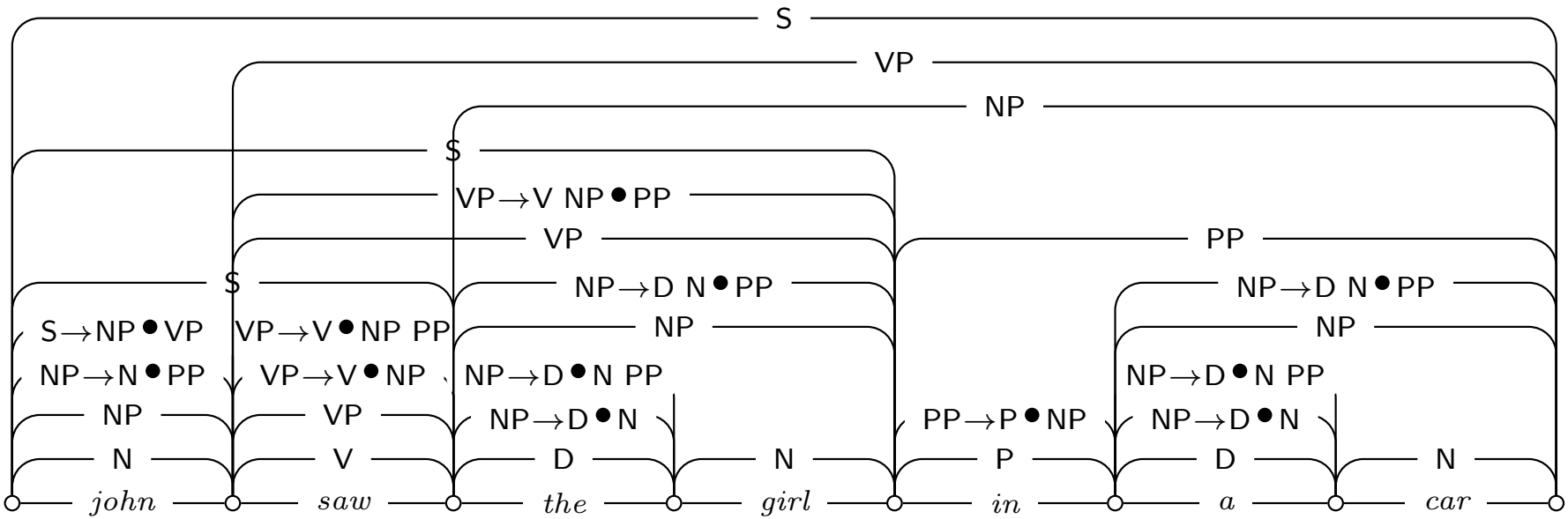
P→ • *in*
PP→ • P NP

NP→ • D N PP
NP→ • D N
NP→ • N PP
NP→ • N
N→ • *john*
N→ • *girl*
N→ • *car*
D→ • *the*
D→ • *a*

N→ • *john*
N→ • *girl*
N→ • *car*

P→ • *in*
PP→ • P NP

Earley Parsing Example



S→ • NP VP
NP→ • D N PP
NP→ • N
NP→ • D N
NP→ • N PP
N→ • *john*
N→ • *girl*
N→ • *car*
D→ • *the*
D→ • *a*

PP→ • P NP
VP→ • V
VP→ • V NP PP
VP→ • V NP
V→ • *saw*

NP→ • D N PP
NP→ • D N
NP→ • N
NP→ • N PP
N→ • *john*
N→ • *girl*
N→ • *car*
D→ • *the*
D→ • *a*

N→ • *john*
N→ • *girl*
N→ • *car*

P→ • *in*
PP→ • P NP

NP→ • D N PP
NP→ • D N
NP→ • N PP
NP→ • N
N→ • *john*
N→ • *girl*
N→ • *car*
D→ • *the*
D→ • *a*

N→ • *john*
N→ • *girl*
N→ • *car*

P→ • *in*
PP→ • P NP

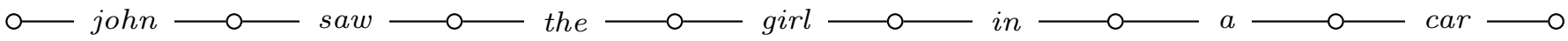
Earley Parsing – Summary

- The number of useless items is reduced
- Superior runtime for unambiguous grammars: $\mathcal{O}(n^2)$
- Valid prefix property
- Not all sub-derivations are computed

Left Corner Parsing

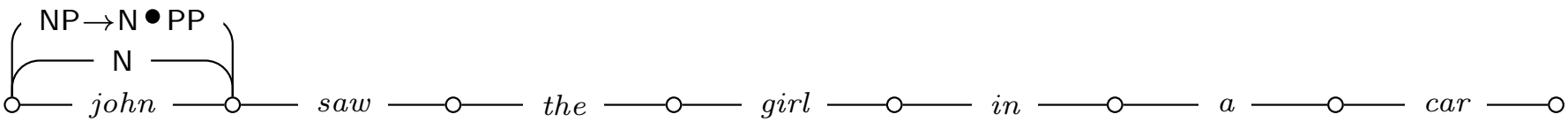
- Observation: Earley parsing predicts items that can not succeed
- Idea: predict only items that can also be derived from the leftmost terminal item
- Formalization: left-corner relation
  - $A >_l B \Longleftrightarrow \exists \beta : A \rightarrow B\beta \in P, B \in \Sigma \cup N$
  - $A >_l^*$ is the transitive closure of $>_l$
- Reformulation of the prediction step:
  - If $(A \rightarrow \beta \bullet Y \alpha, i, j)$ and $(B, j, k) \in \mathcal{C}$, with $B \in \Sigma \cup N$ add $(C \rightarrow B \bullet \gamma, j, k)$ if $Y >_l^* C$
- This formulation also avoids the zero-length predictions with the dot in initial position

Left-Corner Example

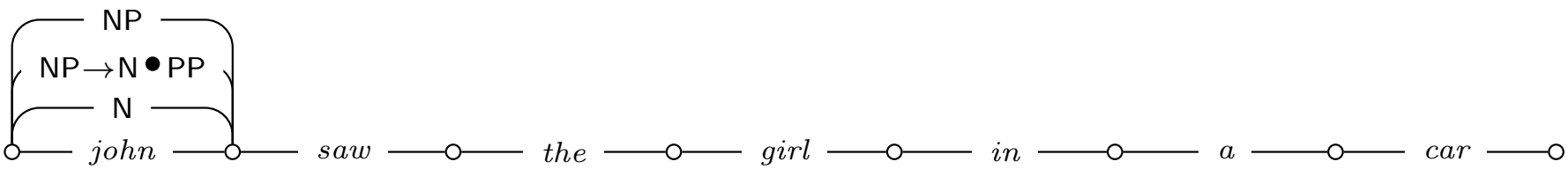


Left-Corner Example

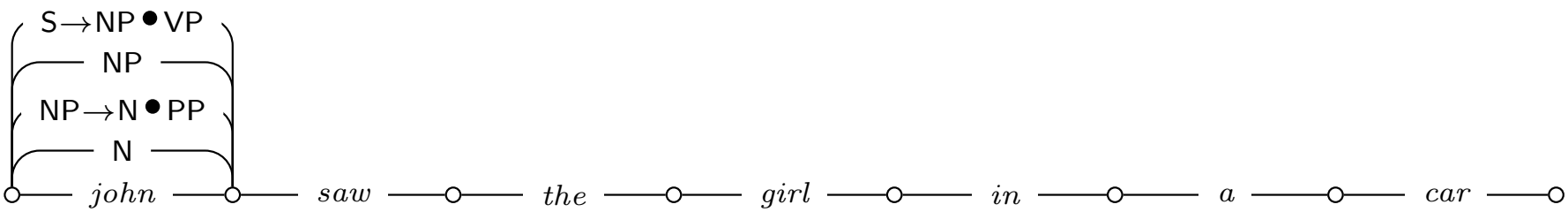
Left-Corner Example



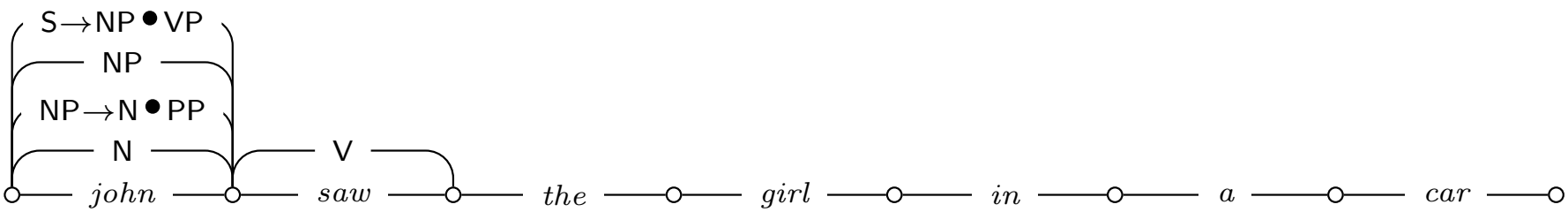
Left-Corner Example



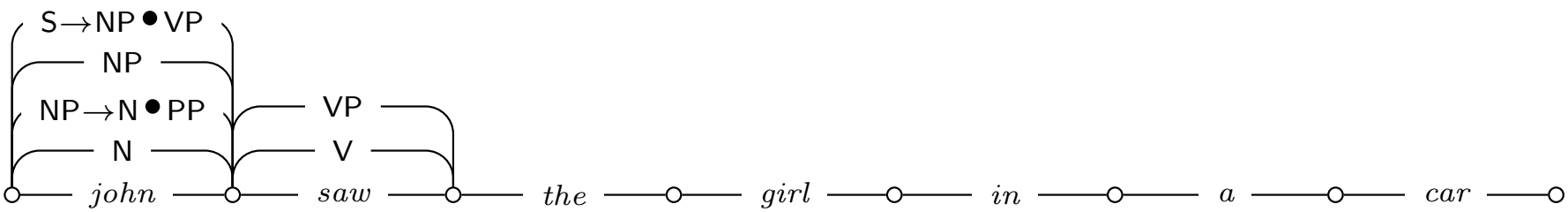
Left-Corner Example



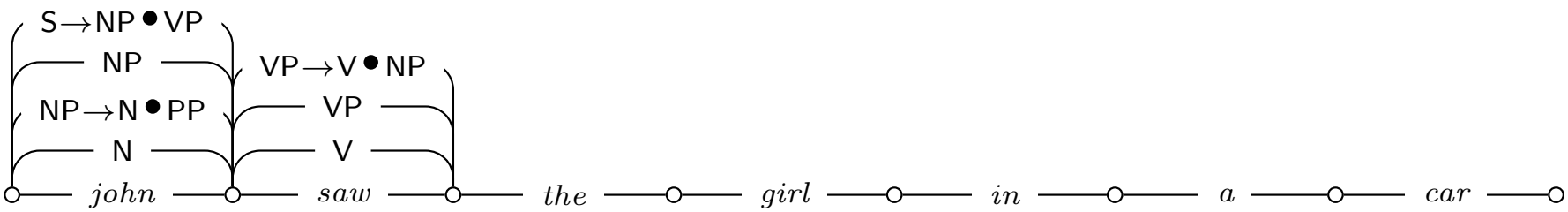
Left-Corner Example



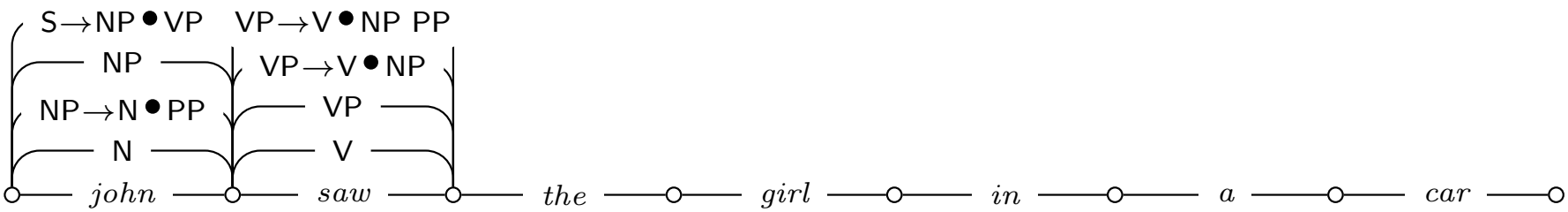
Left-Corner Example



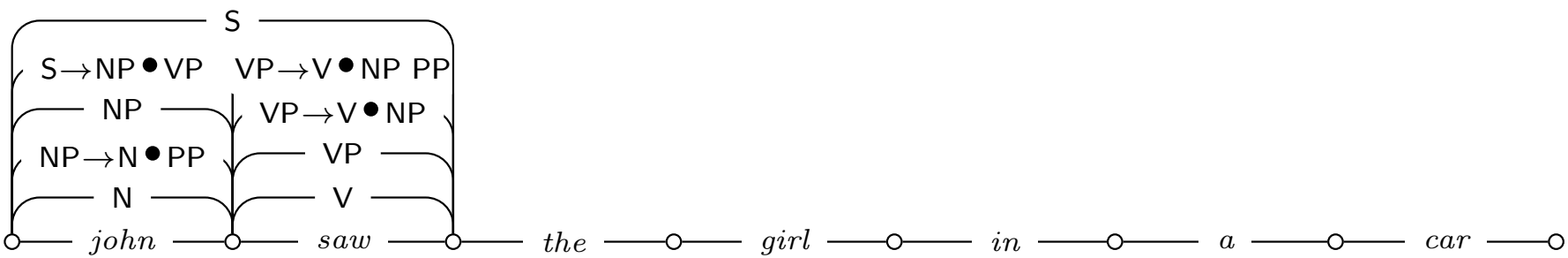
Left-Corner Example



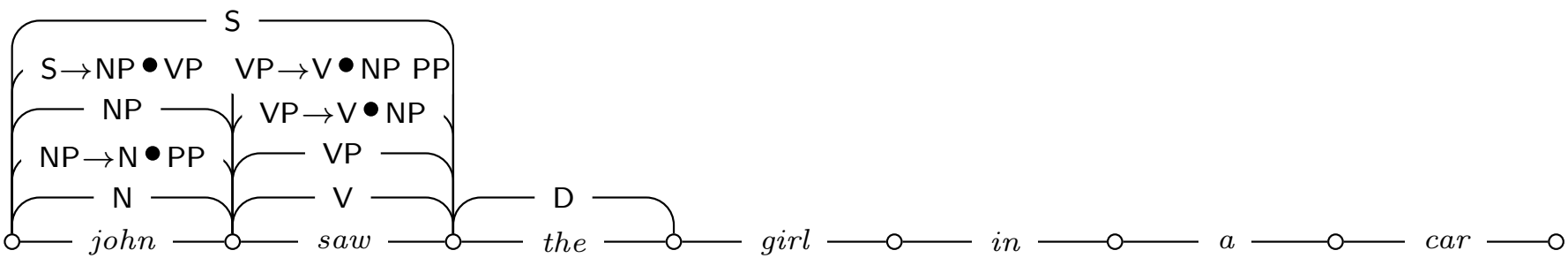
Left-Corner Example



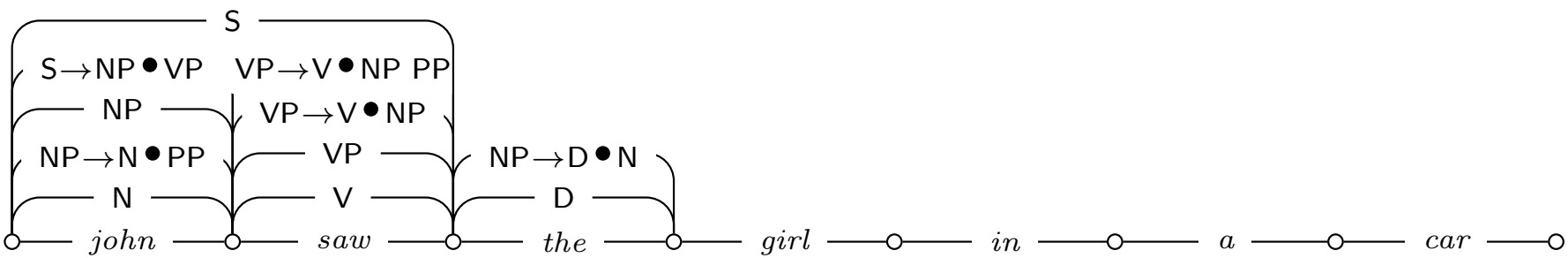
Left-Corner Example



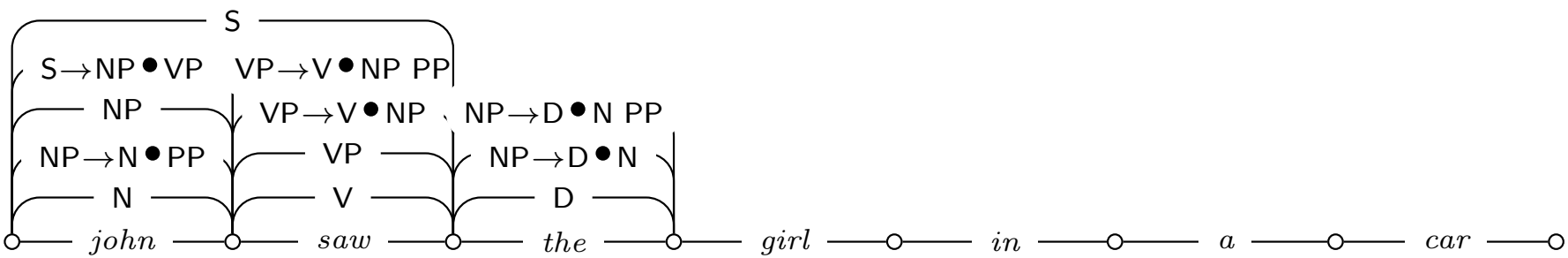
Left-Corner Example



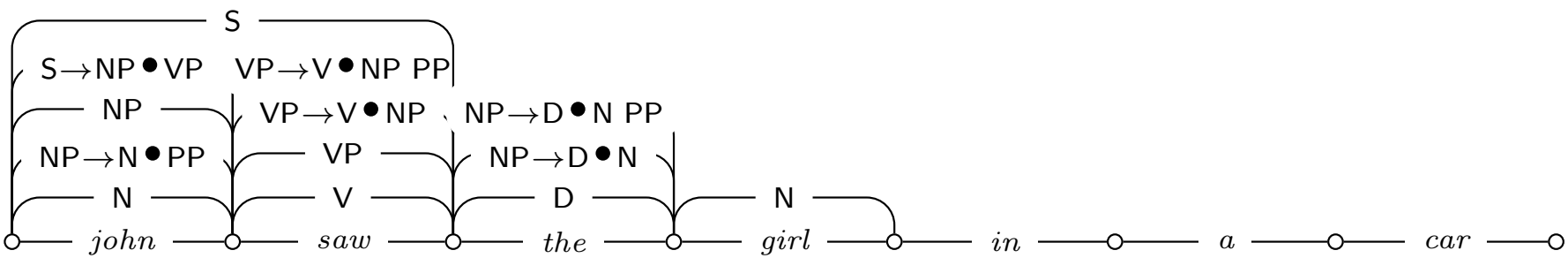
Left-Corner Example



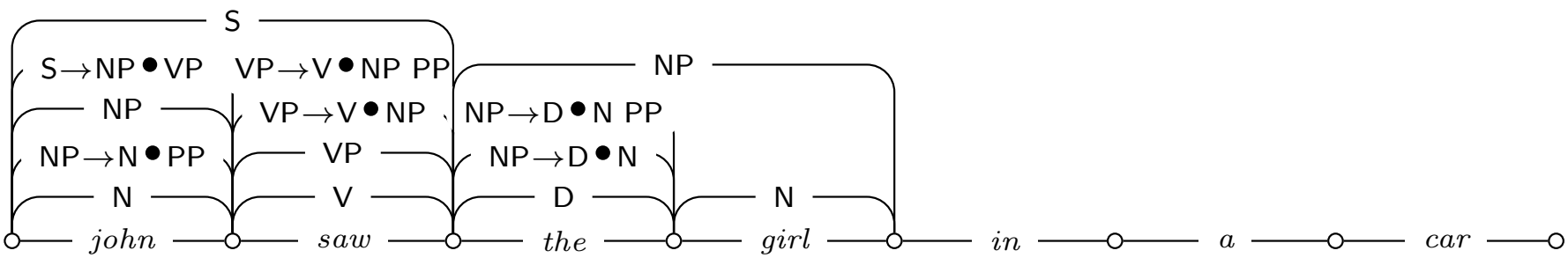
Left-Corner Example



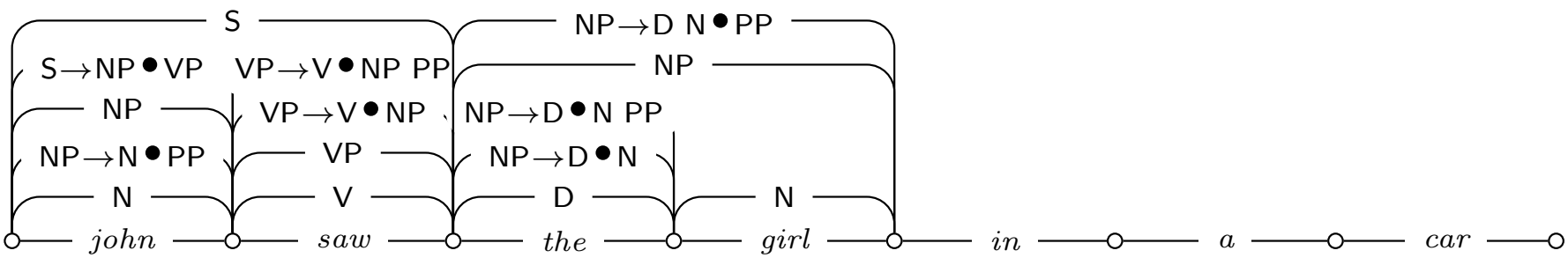
Left-Corner Example



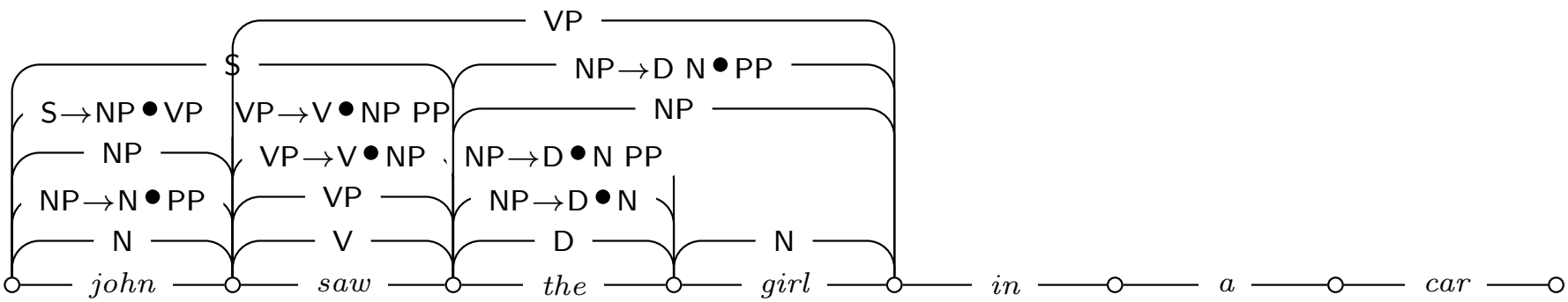
Left-Corner Example



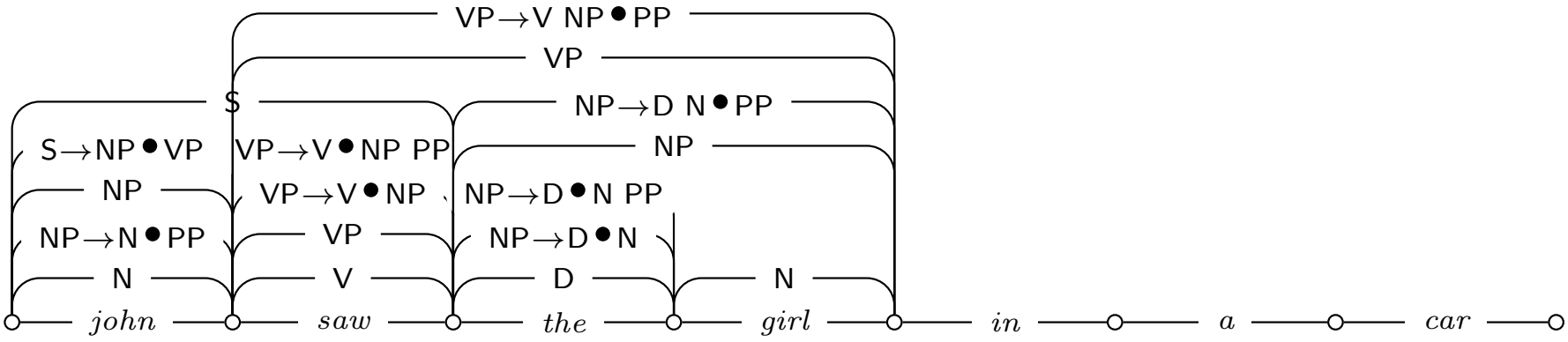
Left-Corner Example



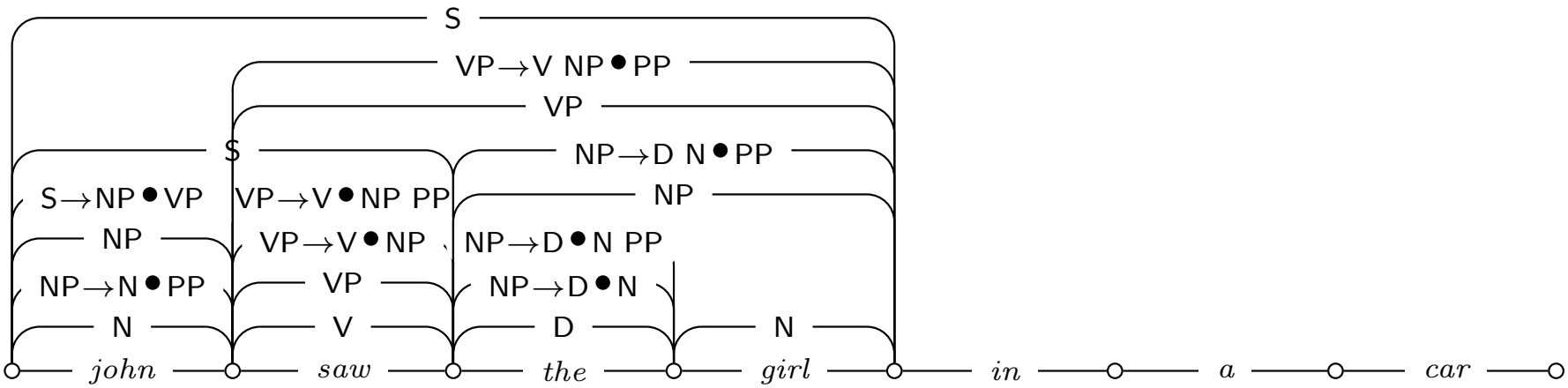
Left-Corner Example



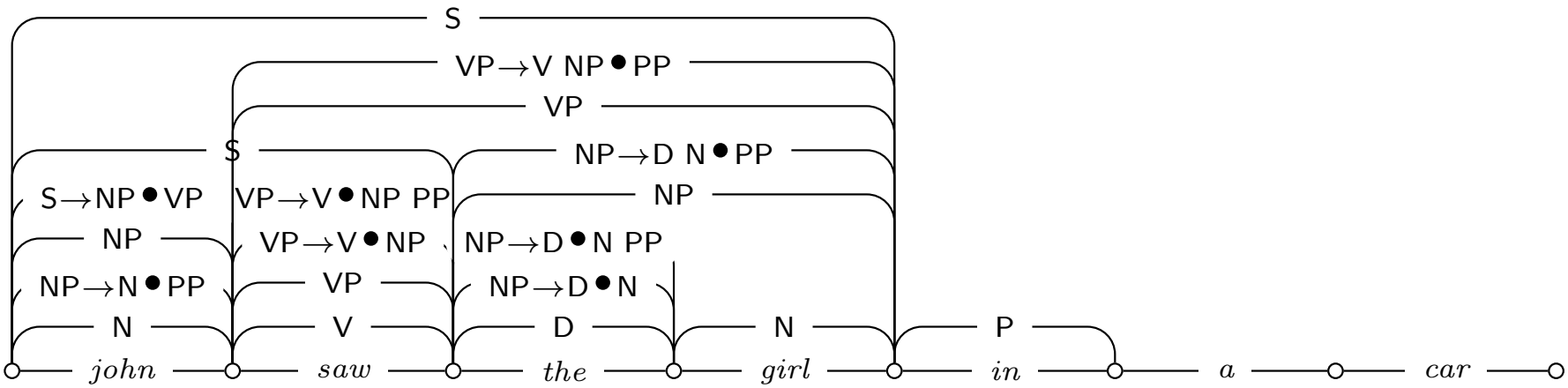
Left-Corner Example



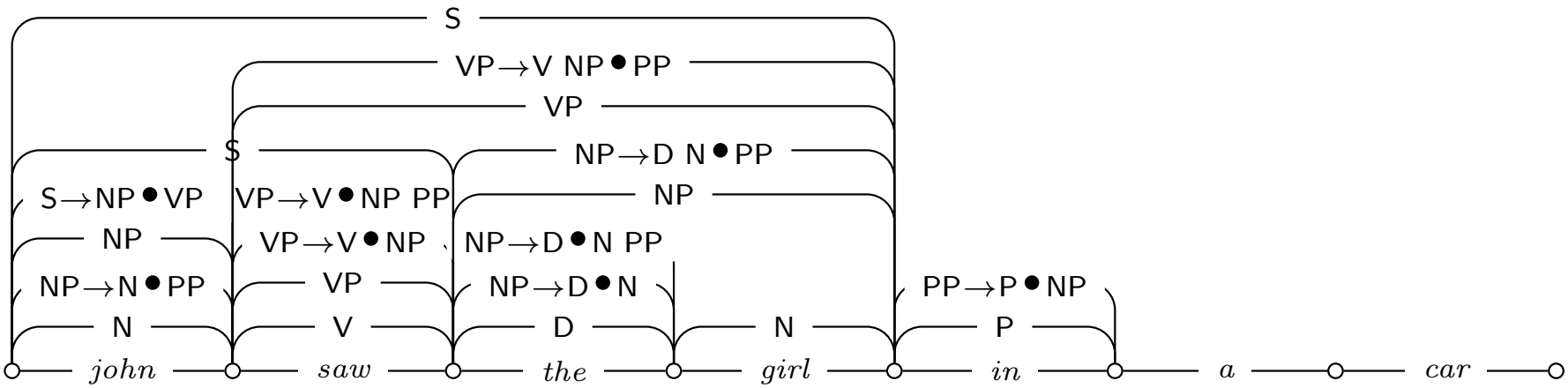
Left-Corner Example



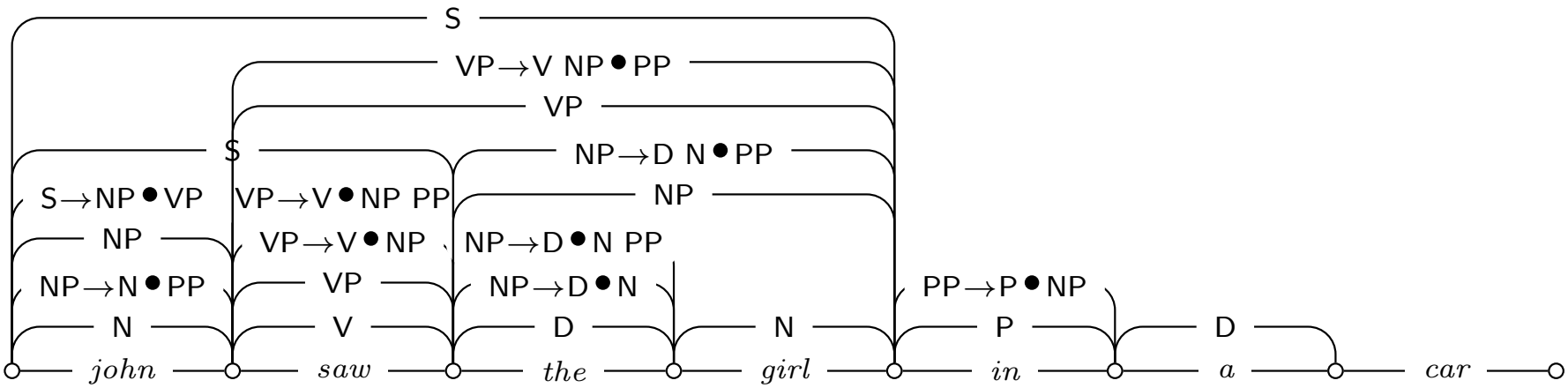
Left-Corner Example



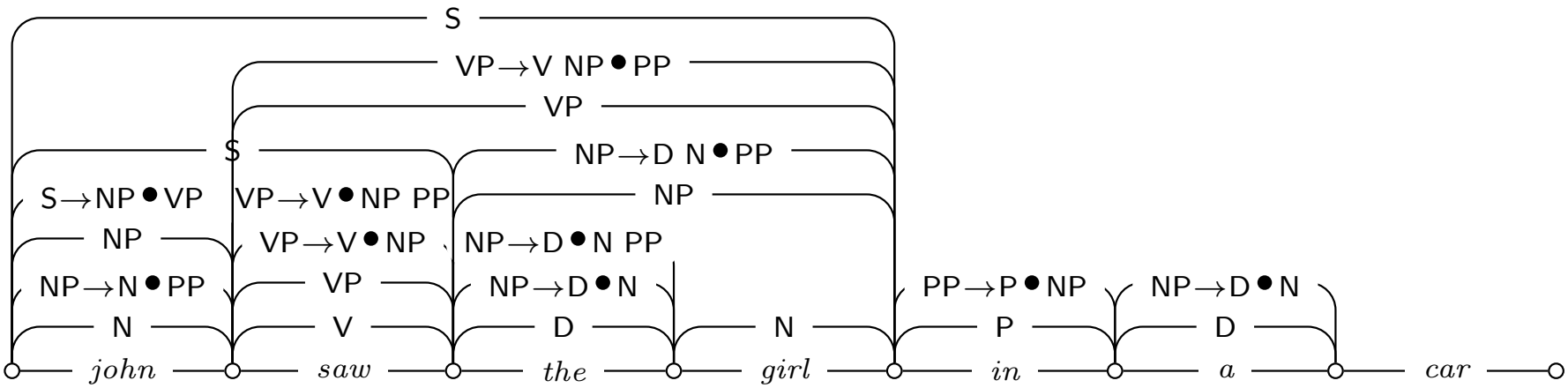
Left-Corner Example



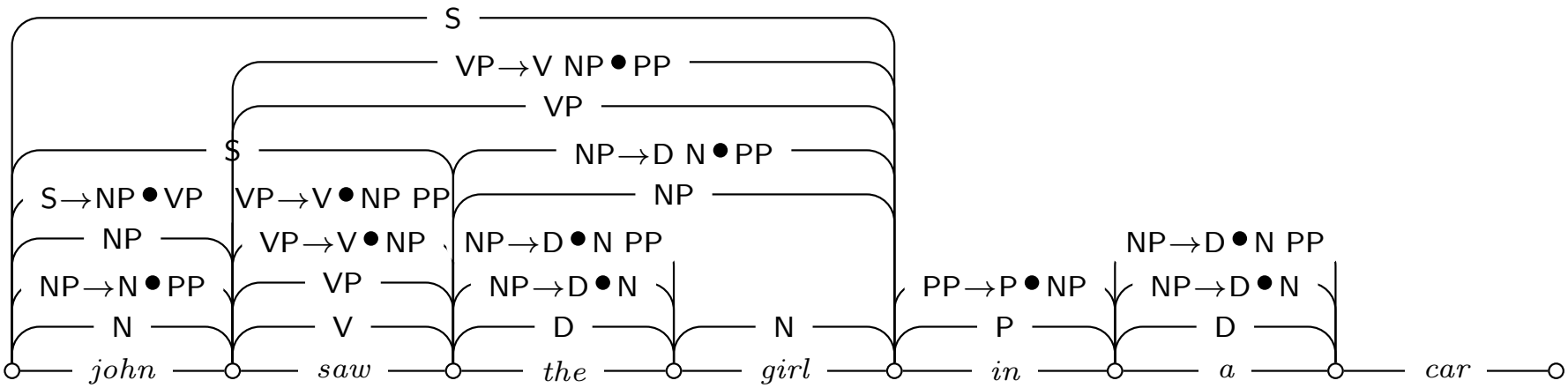
Left-Corner Example



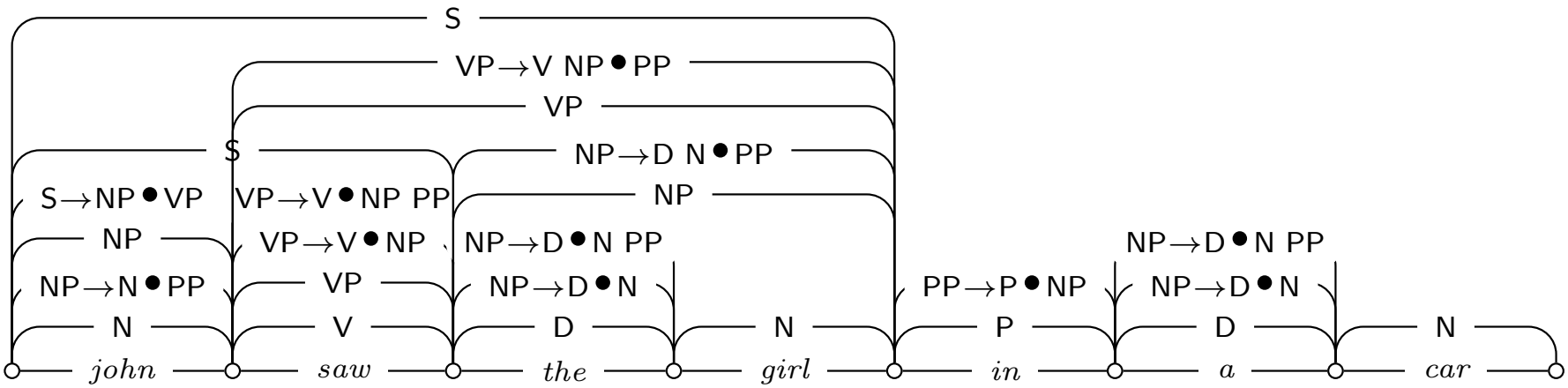
Left-Corner Example



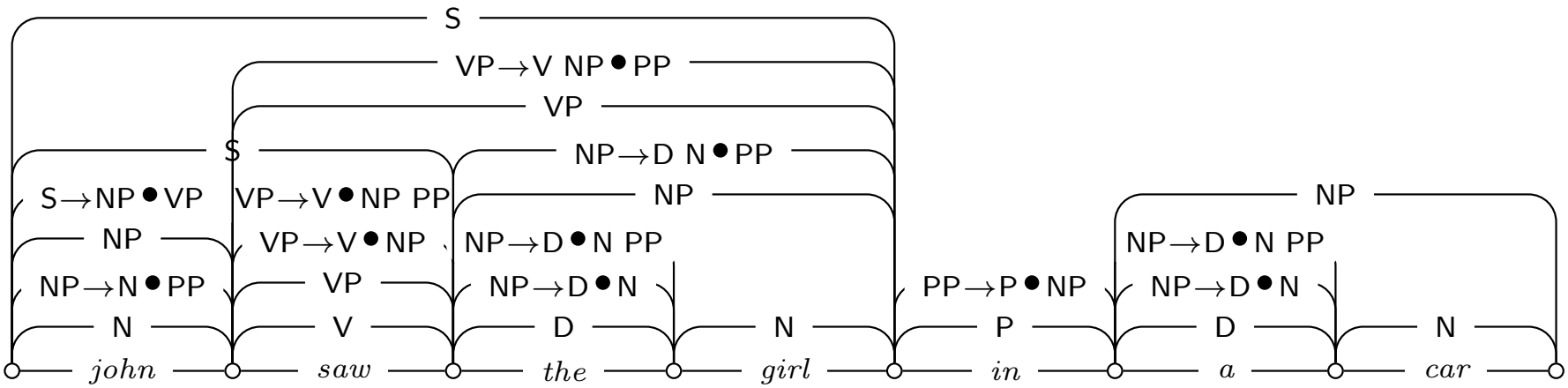
Left-Corner Example



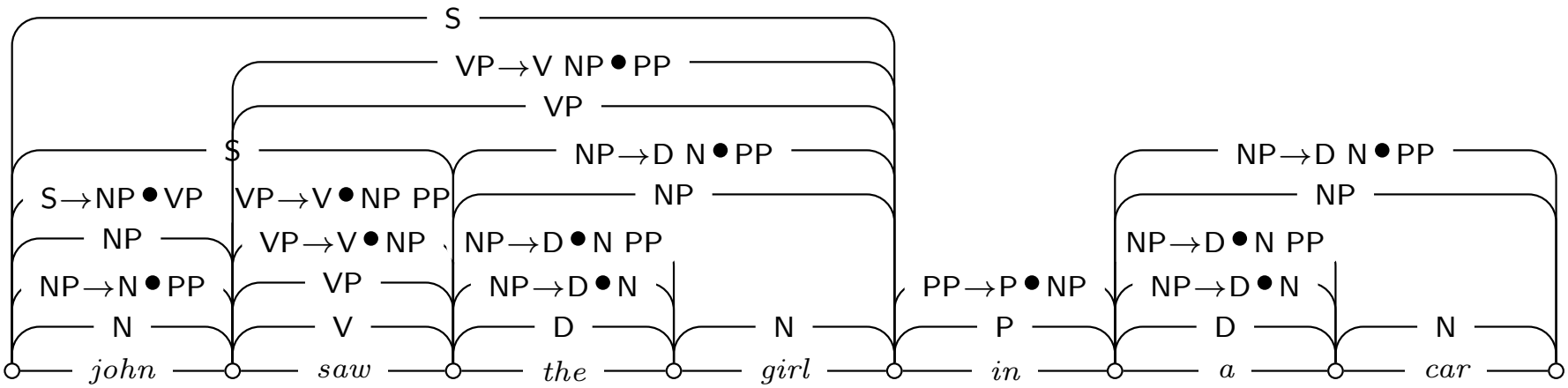
Left-Corner Example



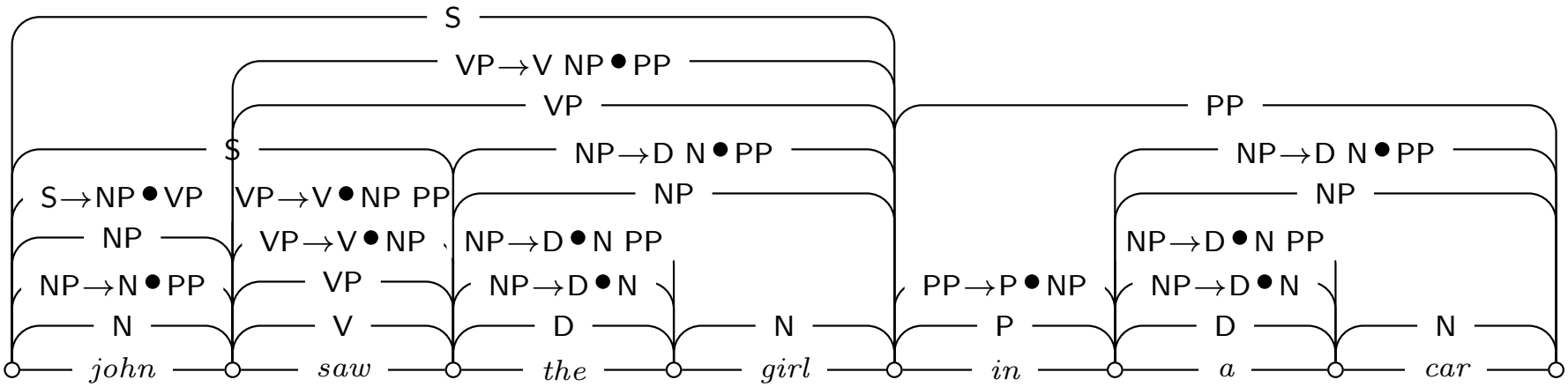
Left-Corner Example



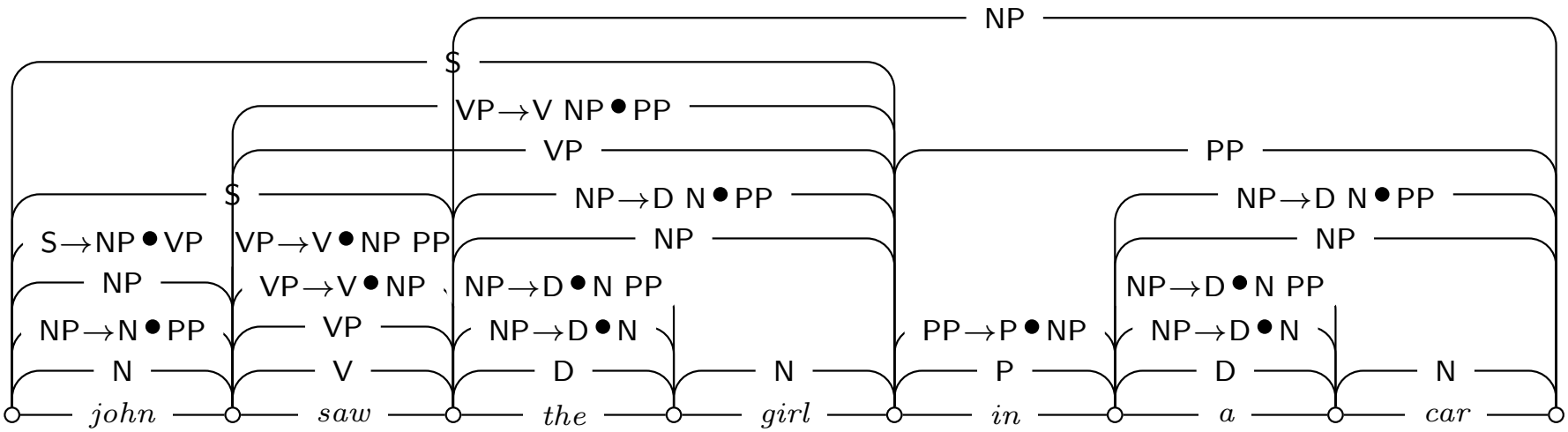
Left-Corner Example



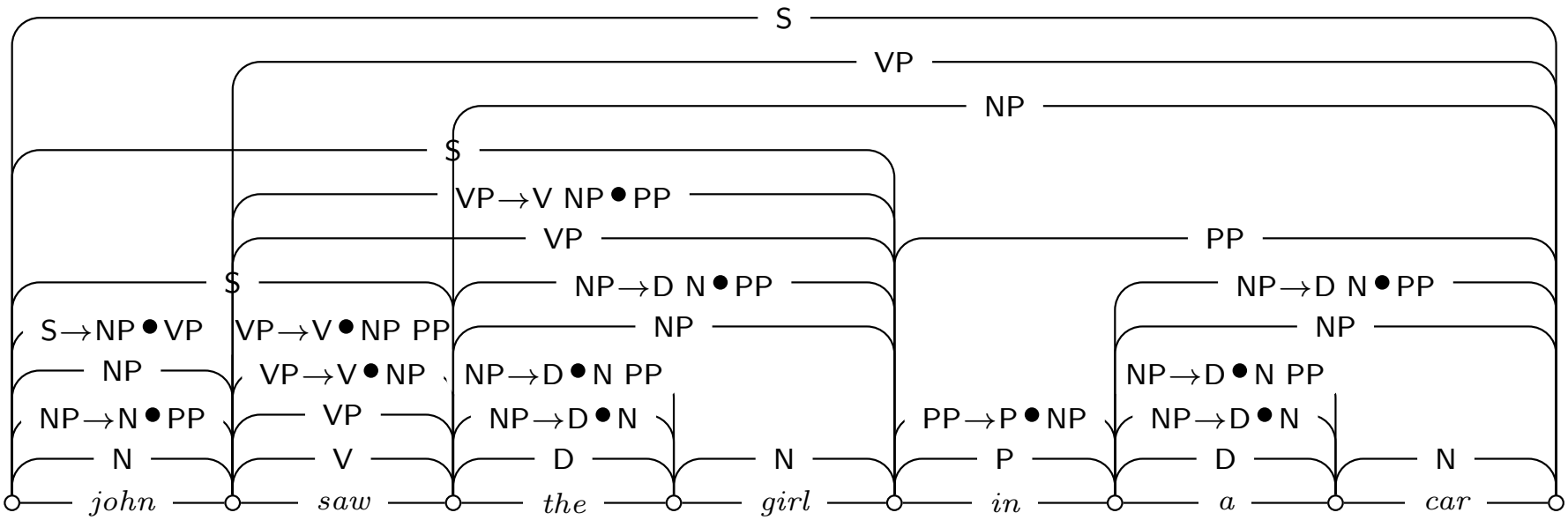
Left-Corner Example



Left-Corner Example



Left-Corner Example



Context-Free Parsing: Summary

- Context-free grammars provide you with a finite set of infinitely embeddable brackets
- Two main approaches to CF recognition: top down (goal-driven) and bottom-up (data driven)
- Storing sub-derivations for re-use (*dynamic programming*) in a chart lead to a polynomial algorithm with worst case n^3
- The chart offers a compact (polynomial size) storage for a possibly exponential number of results
- Earley and Left Corner Parsing improve the average runtime over the naïve CYK algorithm, and have a better worst case complexity for some classes of context-free grammars