CMPU 240 · Theory of Computation

Parsing with Context-free Grammars

2 April 2026

Where are we?

We use CFGs to describe both human languages and programming languages.

So we want to be able to compute whether a given string is in the language of a particular CFG:

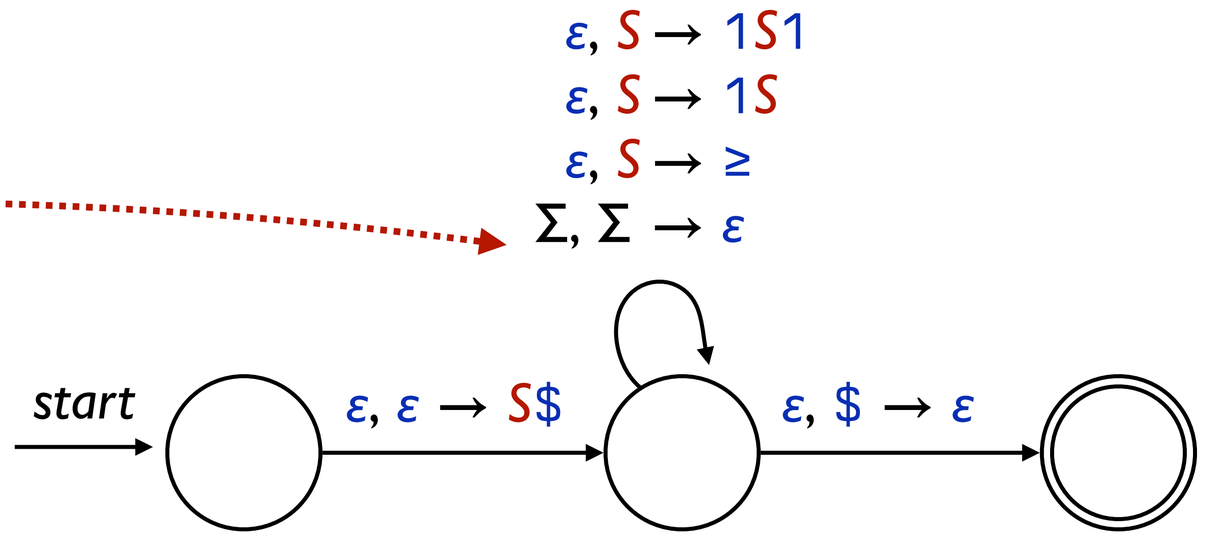
- Is this a well-formed utterance in English?
- Is this a syntactically correct Python program?

Why isn't this good enough?

$S \rightarrow 1S1$

$S \rightarrow 1S$

$S \rightarrow \geq$



Given a context-free grammar *G* and a word *w*, we already have a way to tell if *G* generates *w*…

…using nondeterministic guessing of the right rule to use at each step.

We need to be able to do this *algorithmically* – and we'd like it to be fast!

Two parts:

1. Convert the grammar to a “normal form”
2. Use dynamic programming to efficiently parse the input

Chomsky normal form

The definition of a context-free grammar permits unlimited flexibility in the form of the rule bodies.

Advantage: Flexibility in ***designing*** grammars

Disadvantage: Difficulty in ***analyzing*** grammars

A *normal form* grammar obeys certain restrictions on the form of the rules.

- It can simplify derivations.
- It makes proofs about grammars easier because there are fewer possibilities to consider.
- It eliminates redundancy.

Properties of normal forms:

- Every CFL can be generated by a normal form grammar.
- There are algorithmic transformations to convert any CFG into an equivalent grammar in normal form.

Chomsky normal form (CNF)

Each rule is one of these forms:

- $A \to BC$, where neither B nor C is S
- $A \to a$
- $S \to \varepsilon$

Chomsky normal form (CNF)

Each rule is one of these forms:

$A \rightarrow BC$, where neither B nor C is S

$A \rightarrow a$

$S \rightarrow \varepsilon$

Note: Only the start symbol can produce ε

THEOREM Any context-free language is generated by a context-free grammar in Chomsky normal form.

PROOF IDEA Show how to convert a CFG into CNF.

Conversion requires a sequence of rule additions, deletions, and modifications that leave the set of strings generated by the grammar unchanged.

The details are in the reading – and in an appendix to these slides.
We'll just walk through an example in class.

Example conversion

$T \rightarrow aTb \mid \varepsilon$

1. *Make start symbol non-recursive*
2. *Eliminate all ε-rules (except* $S \rightarrow \varepsilon$*)*
3. *Eliminate all chain/unit rules*
4. *Eliminate useless symbols*
5. *Split rules if necessary*

Example conversion

$S \rightarrow T$

$T \rightarrow aTb \mid \varepsilon$

1. ***Make start symbol non-recursive***
2. *Eliminate all ε-rules (except* $S \rightarrow \varepsilon$*)*
3. *Eliminate all chain/unit rules*
4. *Eliminate useless symbols*
5. *Split rules if necessary*

Example conversion

$S \rightarrow T \mid \varepsilon$

$T \rightarrow aTb \mid ab$

1. *Make start symbol non-recursive*
2. ***Eliminate all ε-rules (except $S \rightarrow \varepsilon$)***
3. *Eliminate all chain/unit rules*
4. *Eliminate useless symbols*
5. *Split rules if necessary*

Example conversion

$S \rightarrow aTb \mid ab \mid \varepsilon$

$T \rightarrow aTb \mid ab$

1. *Make start symbol non-recursive*
2. *Eliminate all ε-rules (except $S \rightarrow \varepsilon$)*
3. ***Eliminate all chain/unit rules***
4. *Eliminate useless symbols*
5. *Split rules if necessary*

Example conversion

$S \to aTb \mid ab \mid \varepsilon$

$T \to aTb \mid ab$

1. *Make start symbol non-recursive*
2. *Eliminate all ε-rules (except $S \to \varepsilon$)*
3. *Eliminate all chain/unit rules*
4. ***Eliminate useless symbols***
5. *Split rules if necessary*

None!

Example conversion

$S \rightarrow V_a T V_b \mid V_a V_b \mid \varepsilon$

$T \rightarrow V_a T V_b \mid V_a V_b$

$V_a \rightarrow \mathrm{a}$

$V_b \rightarrow \mathrm{b}$

1. *Make start symbol non-recursive*
2. *Eliminate all ε-rules (except $S \rightarrow \varepsilon$)*
3. *Eliminate all chain/unit rules*
4. *Eliminate useless symbols*
5. ***Split rules if necessary***

a. Add new rules for terminals

Example conversion

$S \rightarrow V_aC \mid V_aV_b \mid \varepsilon$

$T \rightarrow V_aC \mid V_aV_b$

$C \rightarrow TV_b$

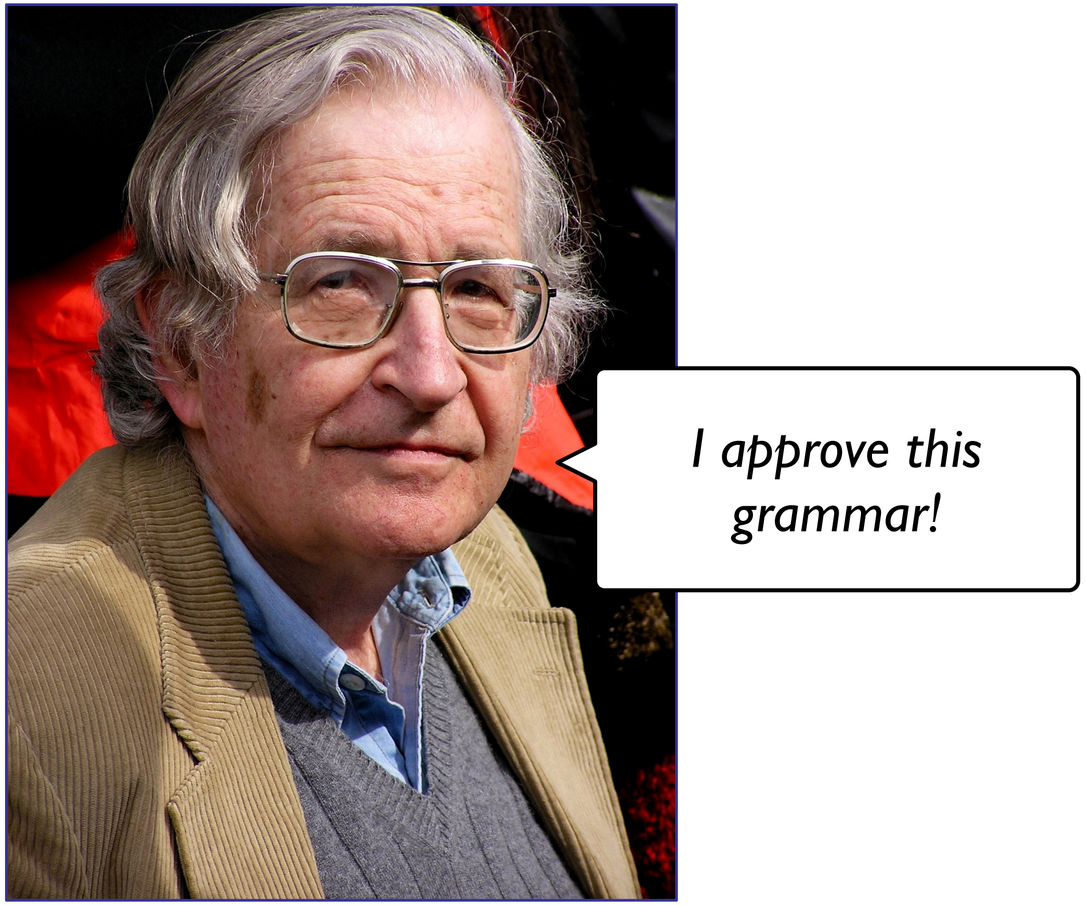
$V_a \rightarrow a$

$V_b \rightarrow b$

1. *Make start symbol non-recursive*
2. *Eliminate all ε-rules (except $S \rightarrow \varepsilon$)*
3. *Eliminate all chain/unit rules*
4. *Eliminate useless symbols*
5. ***Split rules if necessary***

b. Replace rules with three or more symbols on the right side by creating new rules with new variables

And now the grammar is in Chomsky normal form.



CYK parsing algorithm

PROBLEM Given a CFG $G = (V, \Sigma, P, S)$ and a string $w \in \Sigma^*$, determine if w is in $L(G)$, that is, whether $S \overset{*}{\Rightarrow} w$.

The Cocke–Younger–Kasami (CYK) algorithm is an $O(n^3)$ algorithm (n = length of input string) that uses dynamic programming to determine the derivability of the string from the grammar.

PROBLEM Given a CFG $G = (V, \Sigma, P, S)$ and a string $w \in \Sigma^*$, determine if w is in $L(G)$, that is, whether $S \overset{*}{\Rightarrow} w$.

The Cocke–Younger–Kasami (CYK) algorithm is an $O(n^3)$ algorithm (n = length of input string) that uses dynamic programming to determine the derivability of the string from the grammar.

If you haven't seen "Big O Notation" before, it means that it takes at most n^3 steps of computation (loosely defined) to process an input of length n.

PROBLEM Given a CFG $G = (V, \Sigma, P, S)$ and a string $w \in \Sigma^*$, determine if w is in $L(G)$, that is, whether $S \overset{*}{\Rightarrow} w$.

The Cocke–Younger–Kasami (CYK) algorithm is an $O(n^3)$ algorithm (n = length of input string) that uses dynamic programming to determine th
of the string from the grammar.

Dynamic programming is a class of methods that avoid duplicate computation at the expense of memory.

Values that may be used in future computations are stored in a table.

PROBLEM Given a CFG $G = (V, \Sigma, P, S)$ and a string $w \in \Sigma^*$, determine if w is in $L(G)$, that is, whether $S \overset{*}{\Rightarrow} w$.

The Cocke–Younger–Kasami (CYK) algorithm is an $O(n^3)$ algorithm (n = length of input string) that uses dynamic programming to determine the derivability of the string from the grammar.

CYK algorithm

Start with a Chomsky normal form grammar for *L*.

Build a two-dimensional table to keep track of
which portions of the input string can be generated
starting from which variables.

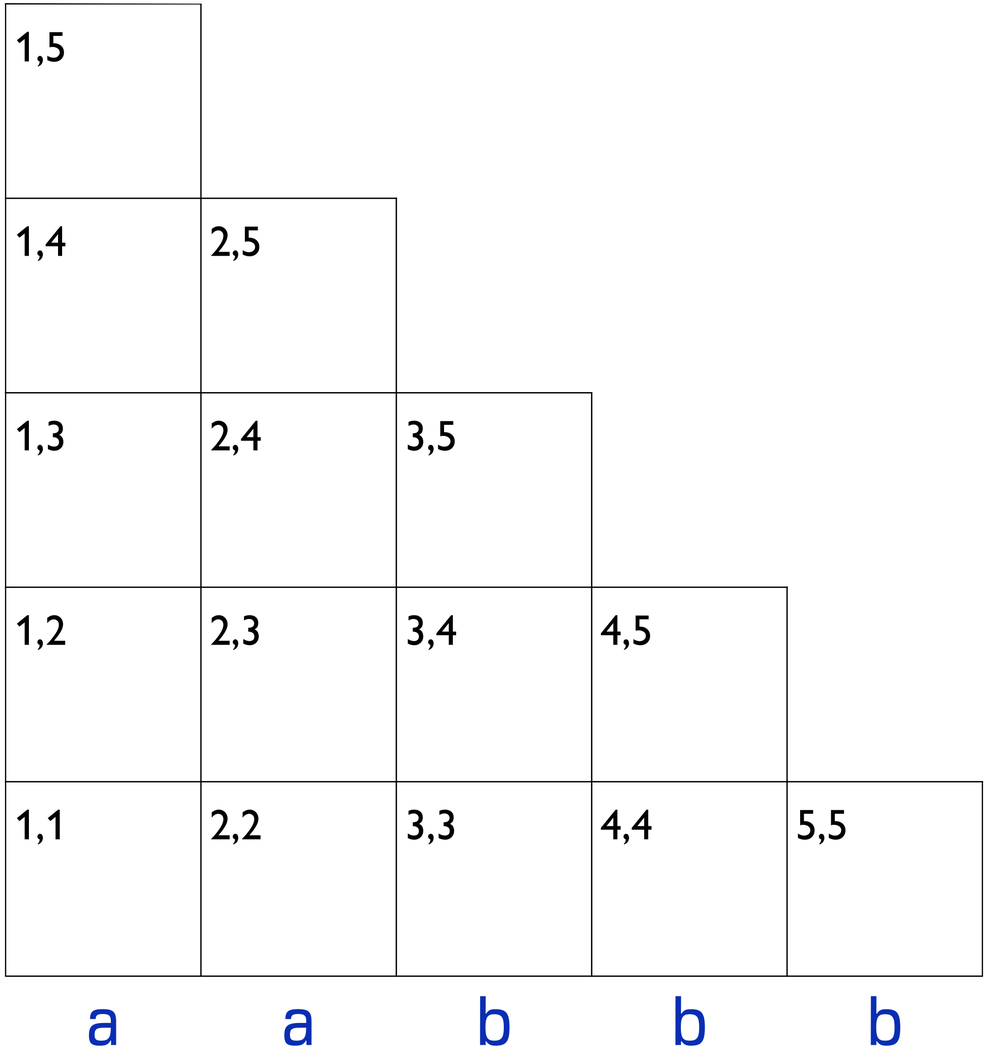
Table

The horizontal axis corresponds to the positions of the string $w = x_1 x_2 \ldots x_n$.

Table entry $X_{i,j}$ is the set of non-terminals A such that

$A \overset{*}{\Rightarrow} w_i\, w_{i+1} \ldots w_j$.

We are particularly interested in whether S is in $X_{1,n}$ because that is the same as saying $S \overset{*}{\Rightarrow} w$ (that is, w is in L)



CYK algorithm strategy

Given a string $w = w_1\, w_2\, w_3 \ldots w_n$, let $w_{i,j}$ be substring $w_i \ldots w_j$ of w.

1. For each substring $w_{i,i}$ of *length 1*, find set $X_{i,i}$ of all variables with rule $A \rightarrow w_{i,i}$.

2. For each substring $w_{i,i+1}$ of *length 2*, find set $X_{i,i+1}$ of all variables that derive $w_{i,i+1}$

…

$n-1$. For substrings $w_{1,n-1}$, $w_{2,n}$ of *length $n-1$*, find sets $X_{1,n-1}$, $X_{2,n}$ of all variables that derive $w_{1,n-1}$, $w_{2,n}$.

n. For the string $w_{1,n} = w$ of *length n*, find the set $X_{1,n}$ of all variables that initiate derivation of w.

At the end, if $S \in X_{1,n}$, then $w \in L(G)$

Example

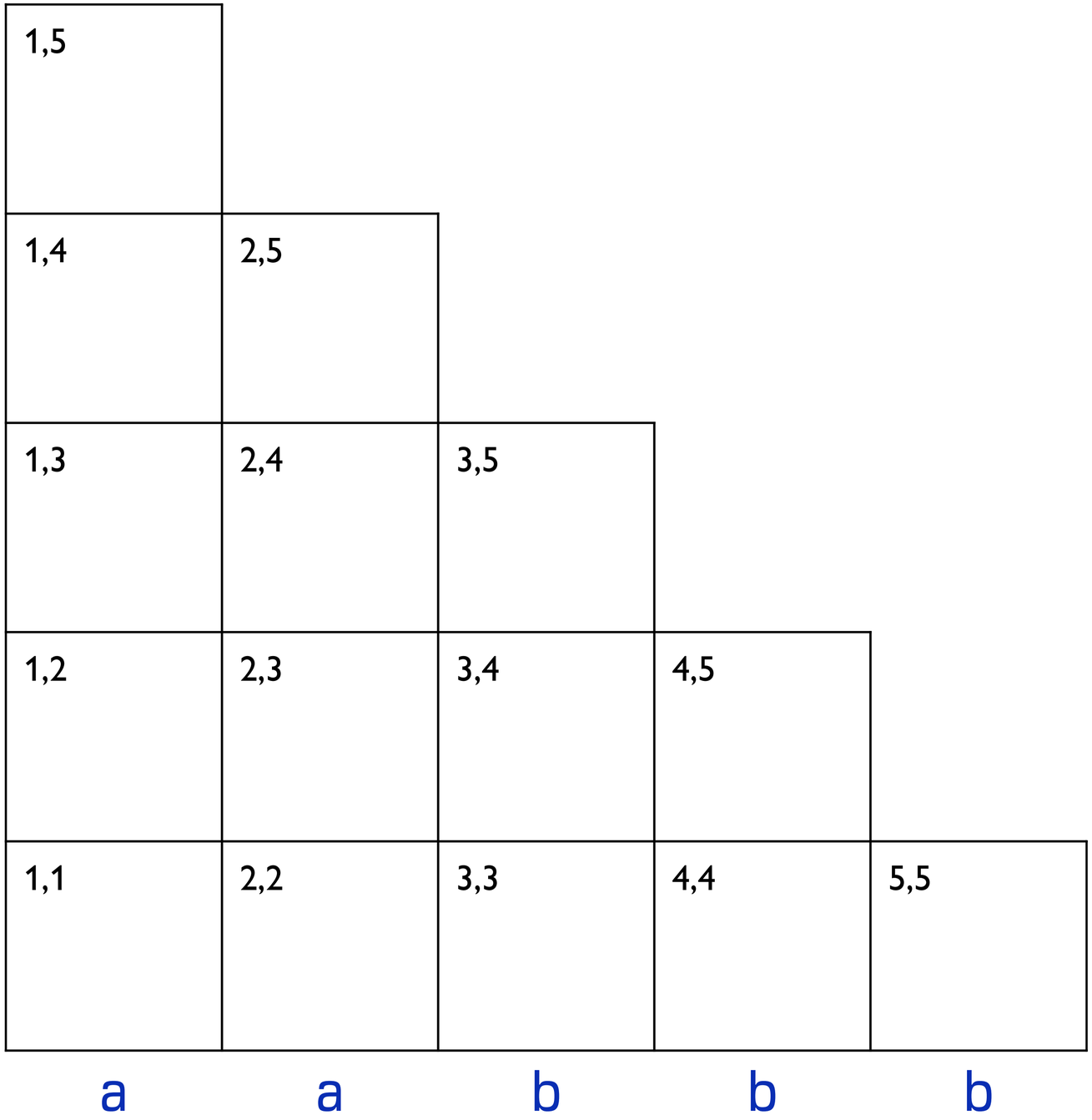
We'll use the algorithm to determine if the string w = aabbb is in the language generated by the grammar

$$S \rightarrow AB$$
$$A \rightarrow BB \mid a$$
$$B \rightarrow AB \mid b$$

$S \rightarrow AB$

$A \rightarrow BB \mid a$

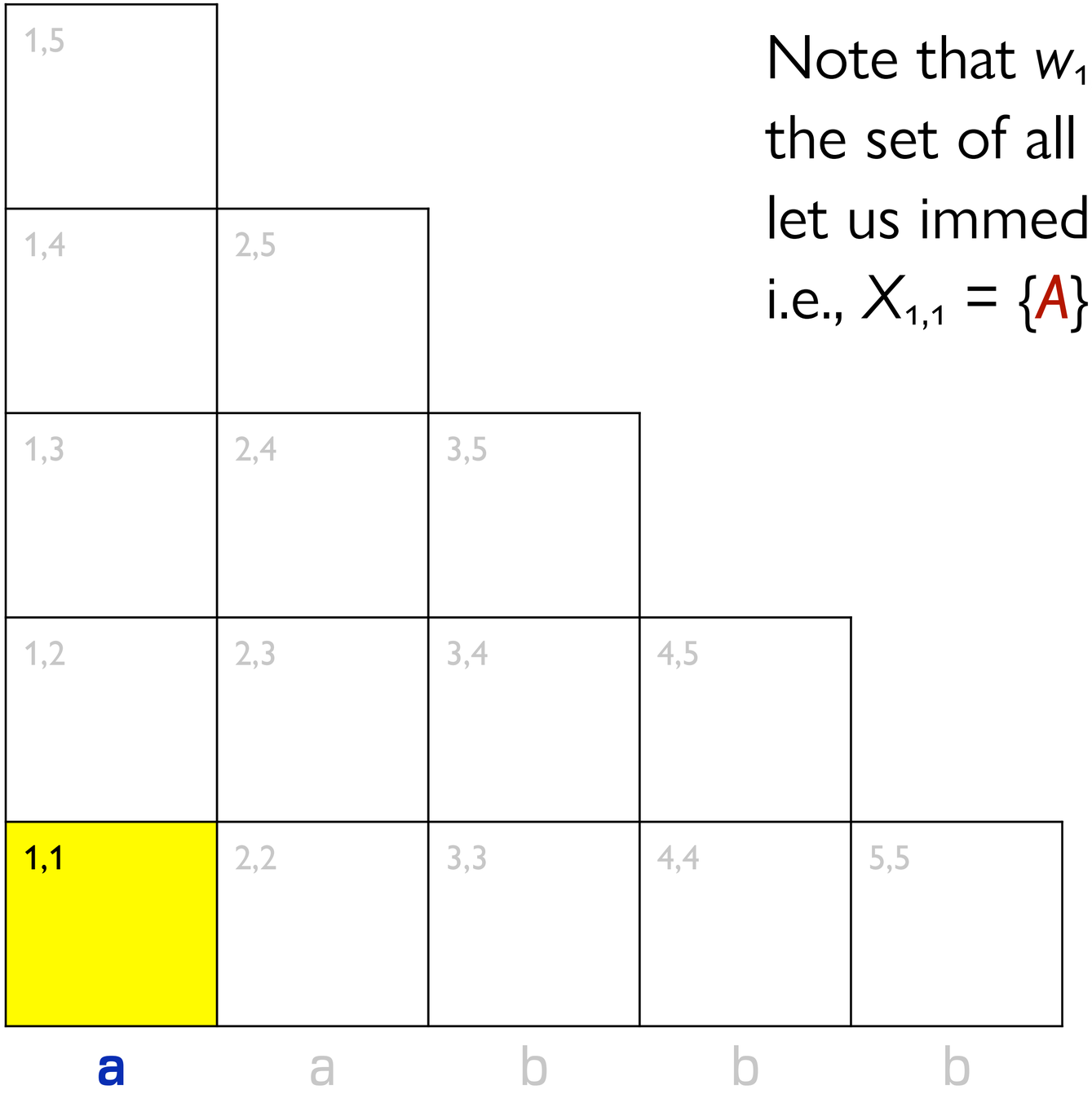
$B \rightarrow AB \mid b$



$S \rightarrow AB$

$A \rightarrow BB \mid a$

$B \rightarrow AB \mid b$

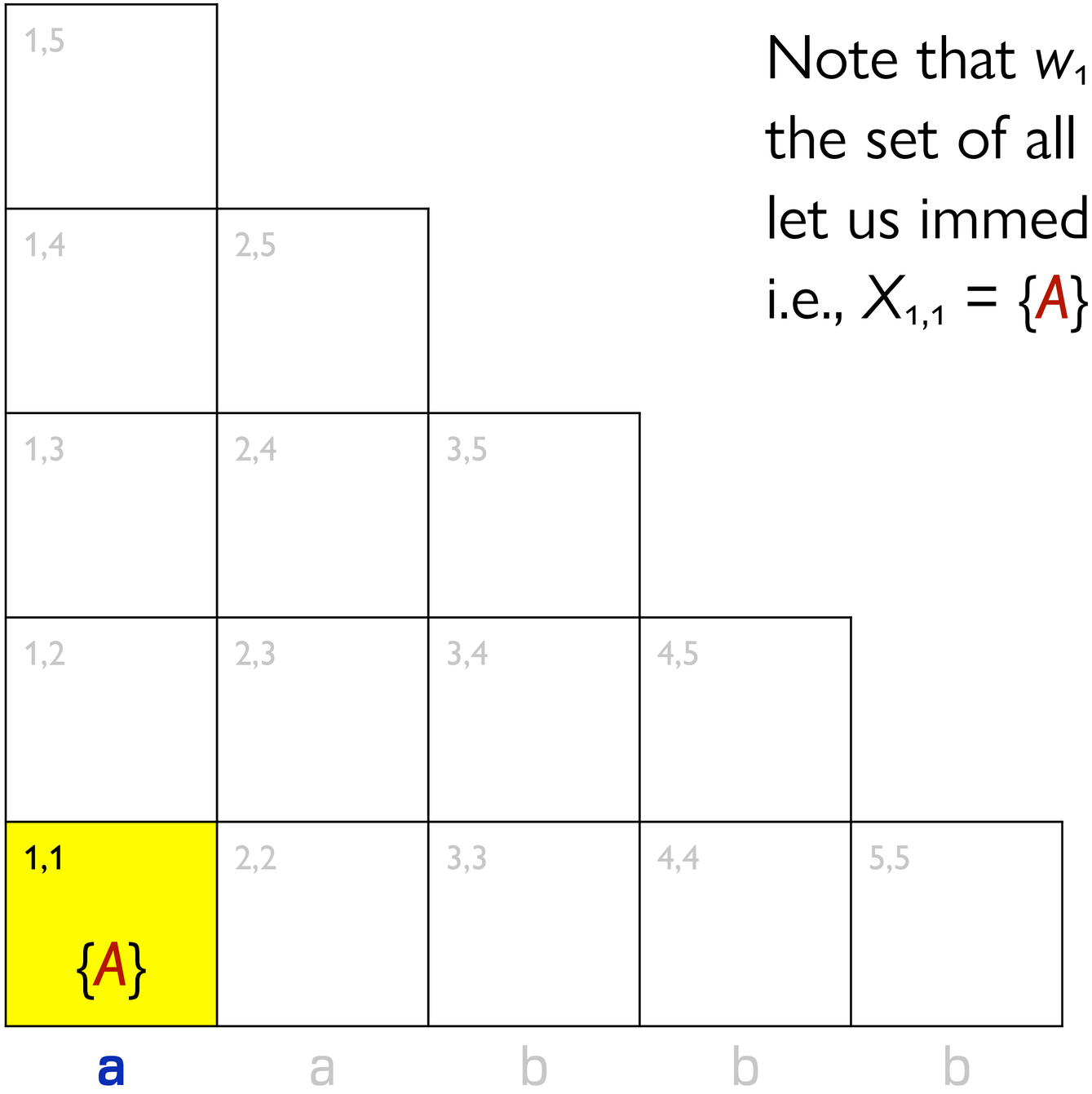


Note that $w_{1,1}$ = a, so $X_{1,1}$ is the set of all variables that let us immediately derive a, i.e., $X_{1,1} = \{A\}$.

$S \rightarrow AB$

$A \rightarrow BB \mid a$

$B \rightarrow AB \mid b$

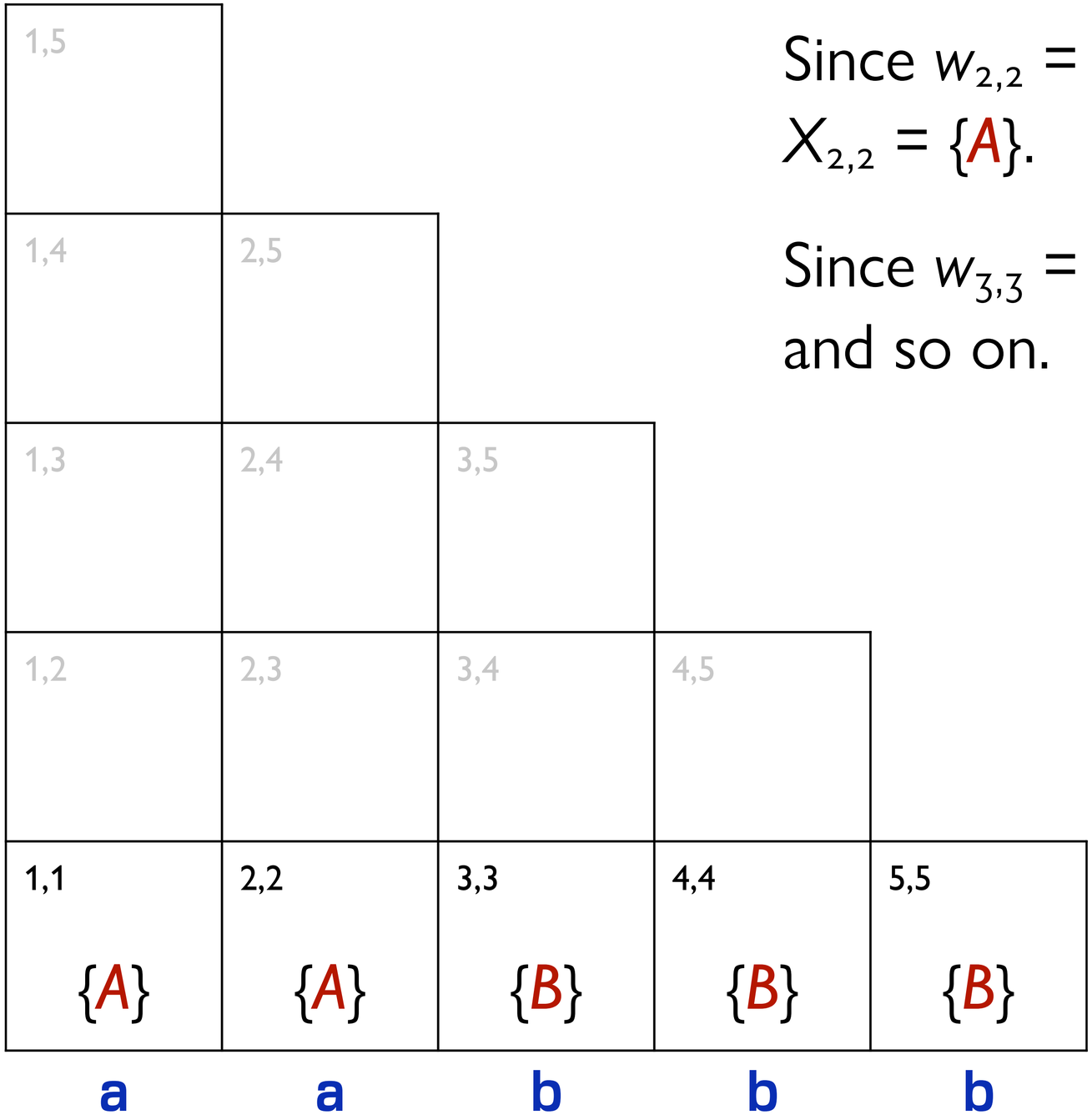


Note that $w_{1,1}$ = a, so $X_{1,1}$ is the set of all variables that let us immediately derive a, i.e., $X_{1,1} = \{A\}$.

$S \rightarrow AB$

$A \rightarrow BB \mid a$

$B \rightarrow AB \mid b$



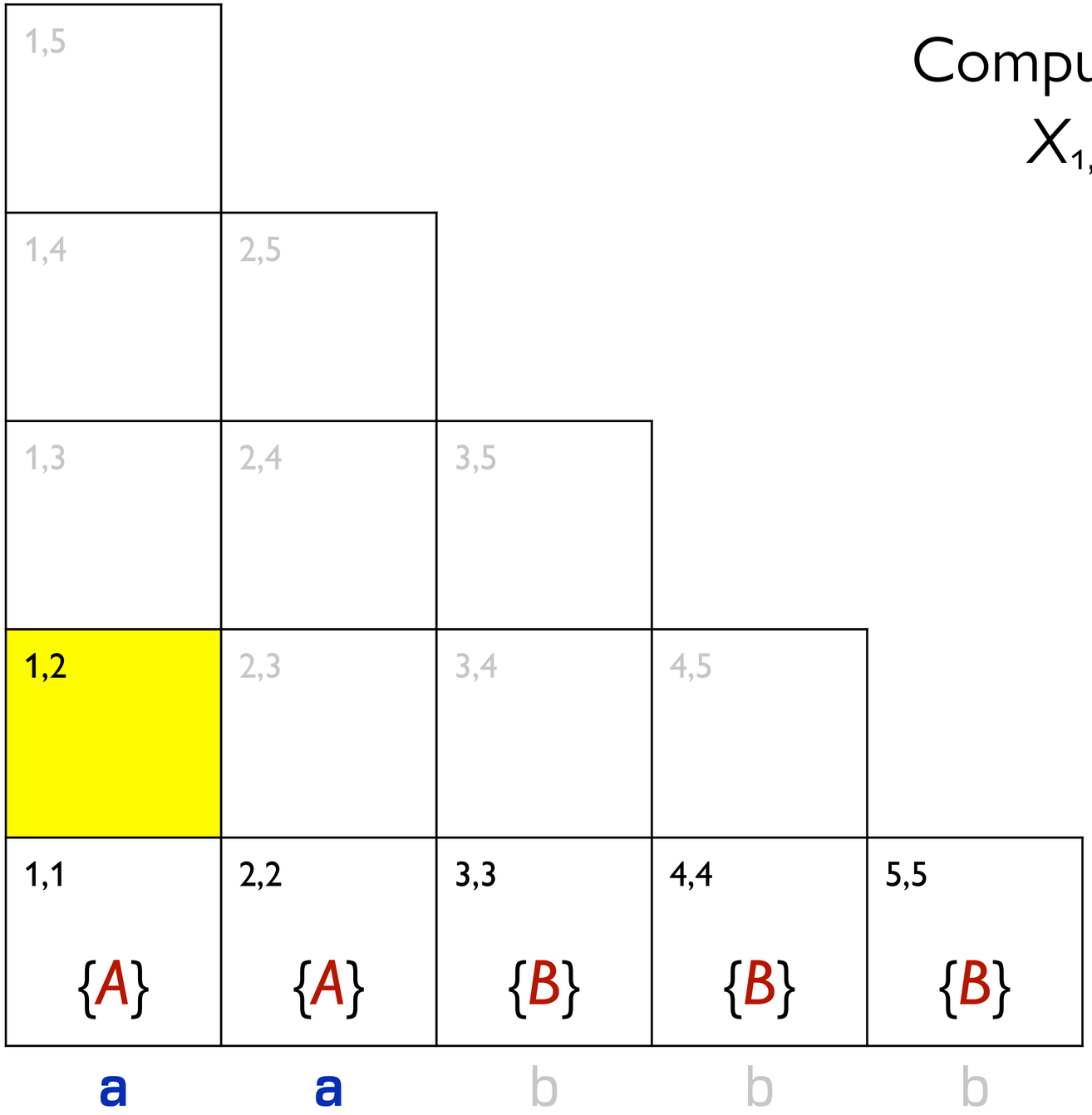
Since $w_{2,2}$ = a, we also have $X_{2,2} = \{A\}$.

Since $w_{3,3}$ = b, $X_{3,3} = \{B\}$, and so on.

$S \to AB$

$A \to BB \mid a$

$B \to AB \mid b$



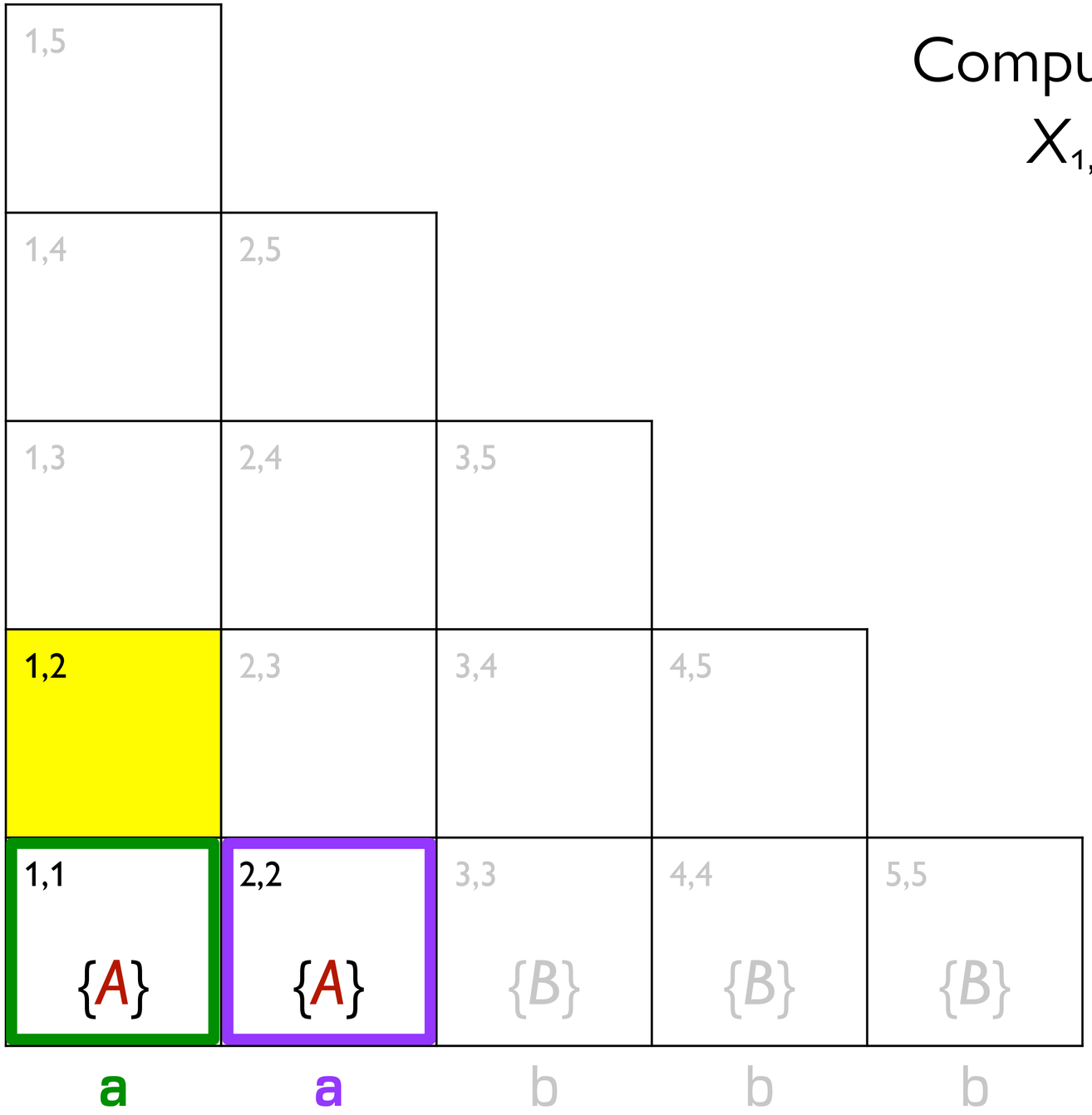
Compute

$$X_{1,2} = \{P \mid P \to QR, Q \in X_{1,1}, R \in X_{2,2}\}.$$

$S \to AB$

$A \to BB \mid a$

$B \to AB \mid b$



Compute

$X_{1,2} = \{P \mid P \to QR,$

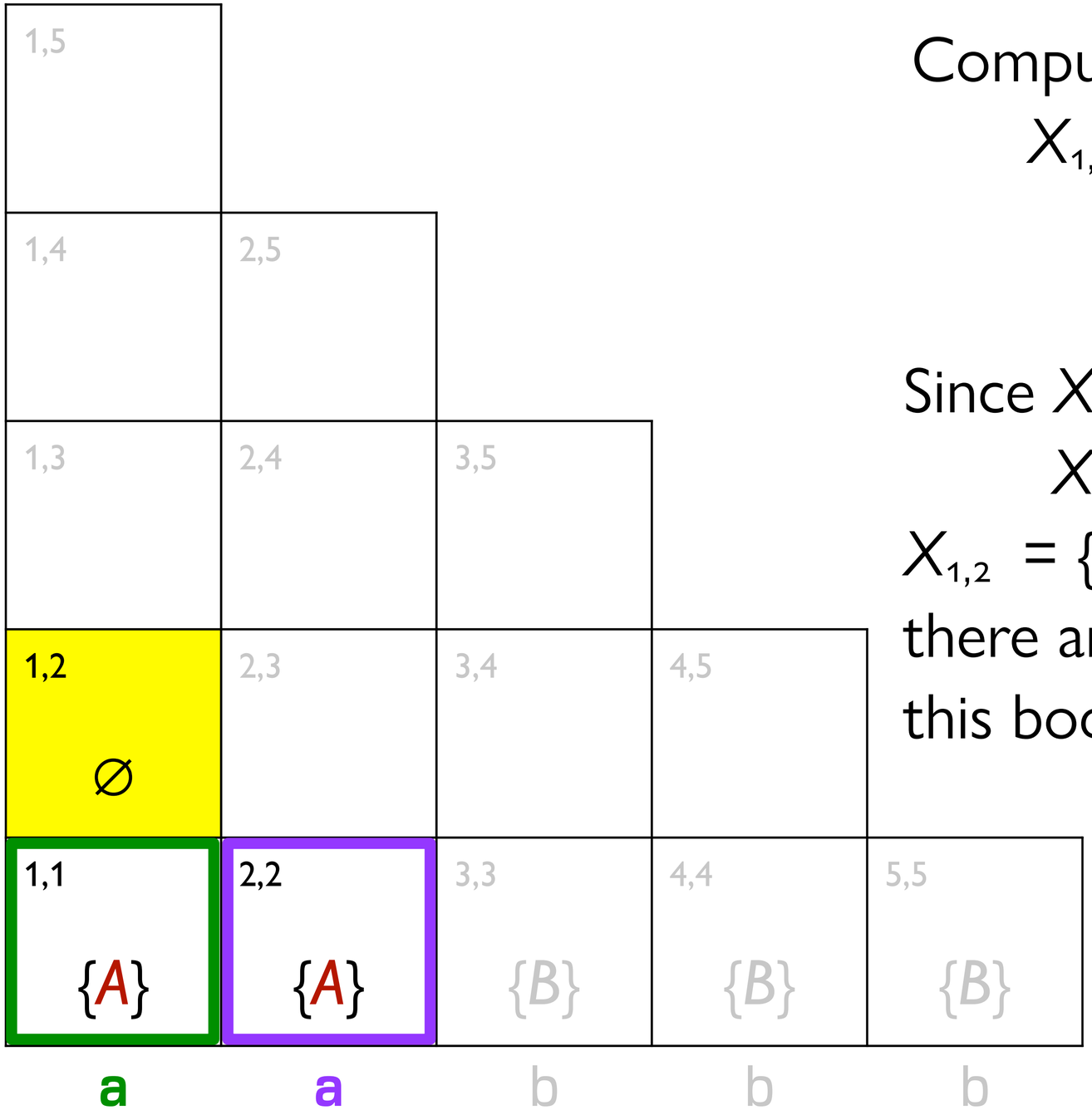
$Q \in X_{1,1},$

$R \in X_{2,2}\}.$

$S \to AB$

$A \to BB \mid a$

$B \to AB \mid b$



Compute

$$X_{1,2} = \{P \mid P \to QR, Q \in X_{1,1}, R \in X_{2,2}\}.$$

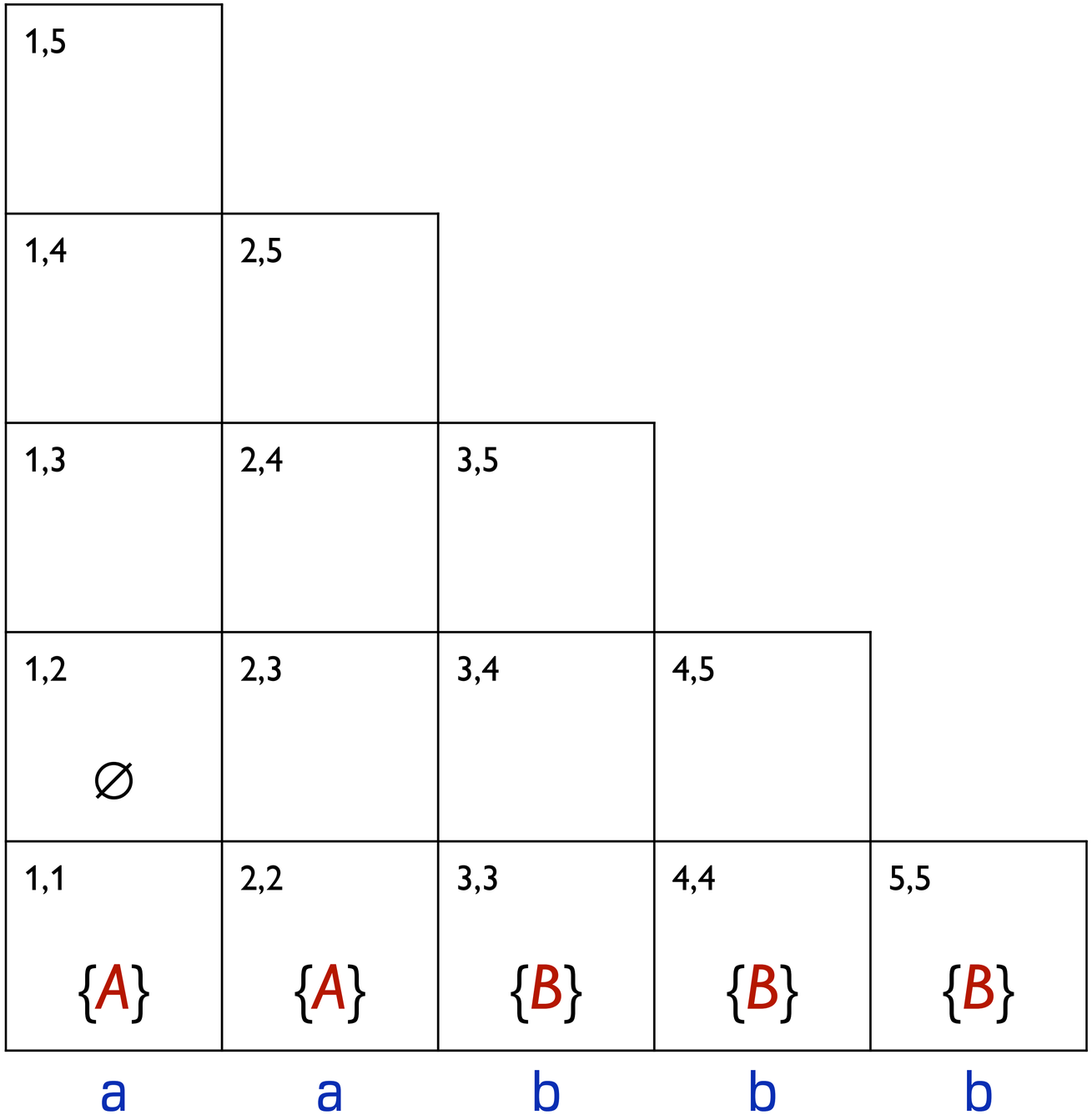
Since $X_{1,1} = \{A\}$ and $X_{2,2} = \{A\}$,

$X_{1,2} = \{P \mid P \to AA\}$, and there are no rules with this body.

$S \rightarrow AB$

$A \rightarrow BB \mid a$

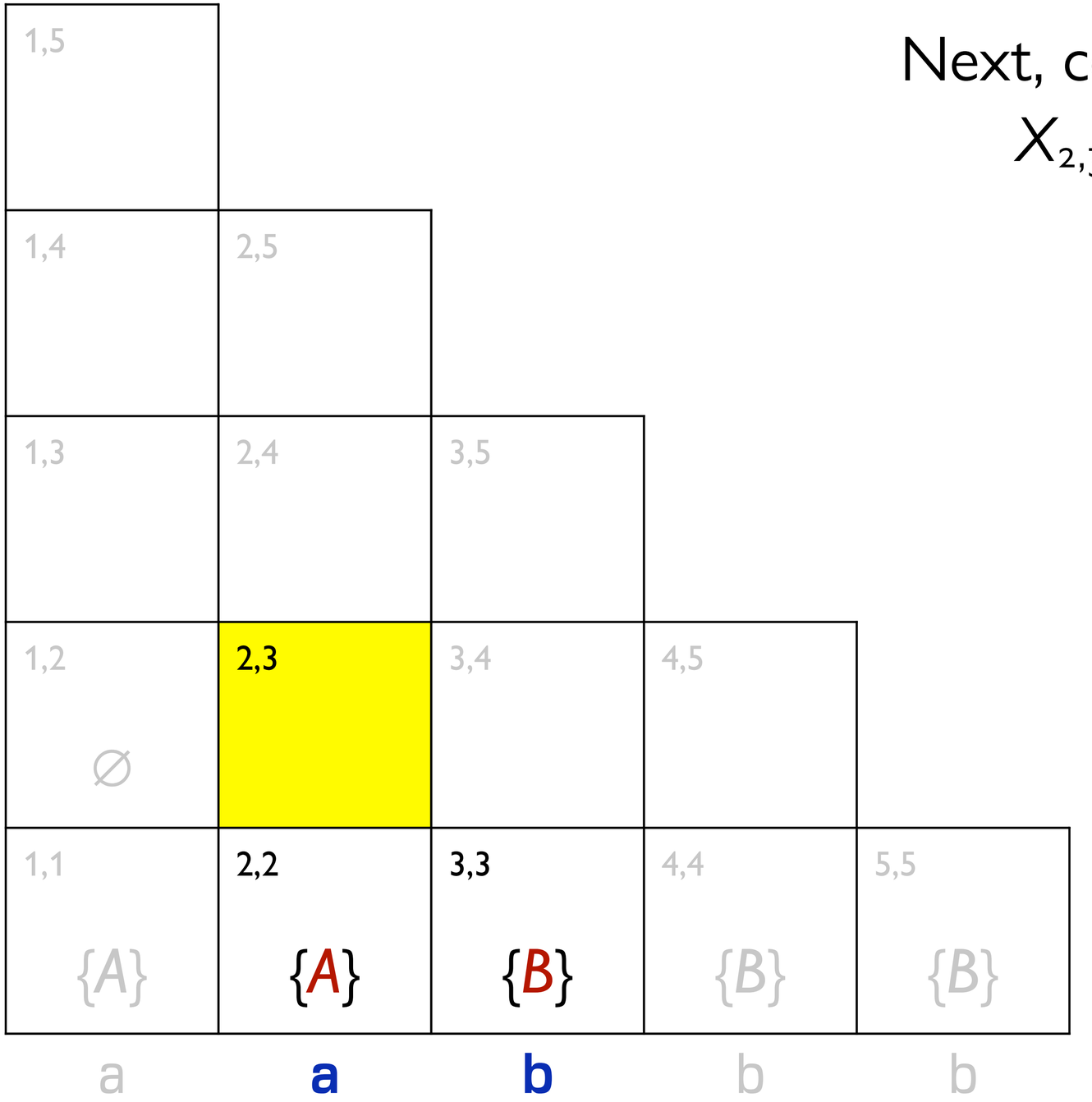
$B \rightarrow AB \mid b$



$S \rightarrow AB$

$A \rightarrow BB \mid a$

$B \rightarrow AB \mid b$



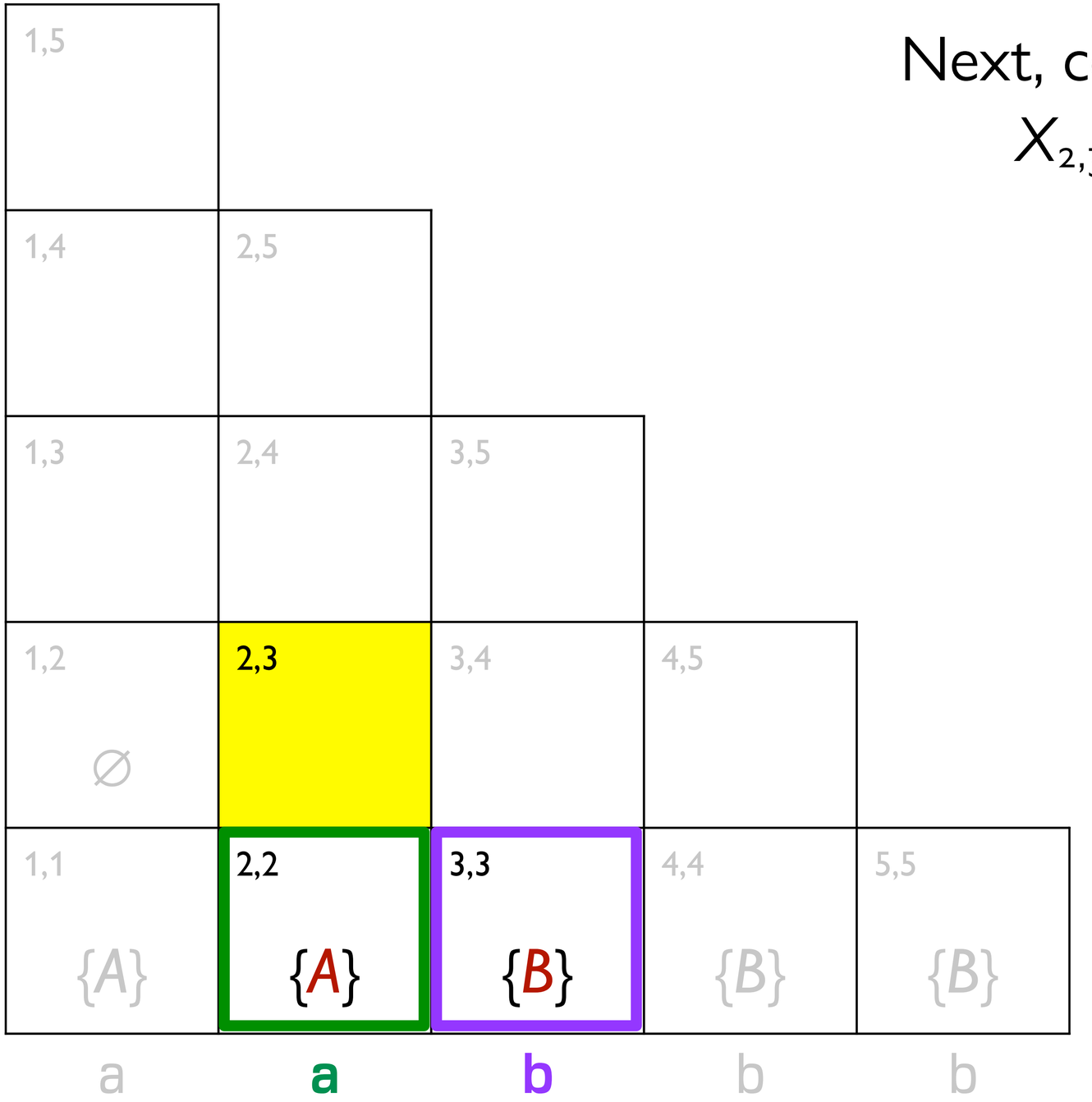
Next, compute

$X_{2,3} = \cdots$

$S \to AB$

$A \to BB \mid a$

$B \to AB \mid b$



Next, compute

$X_{2,3} = \{P \mid P \to QR,$

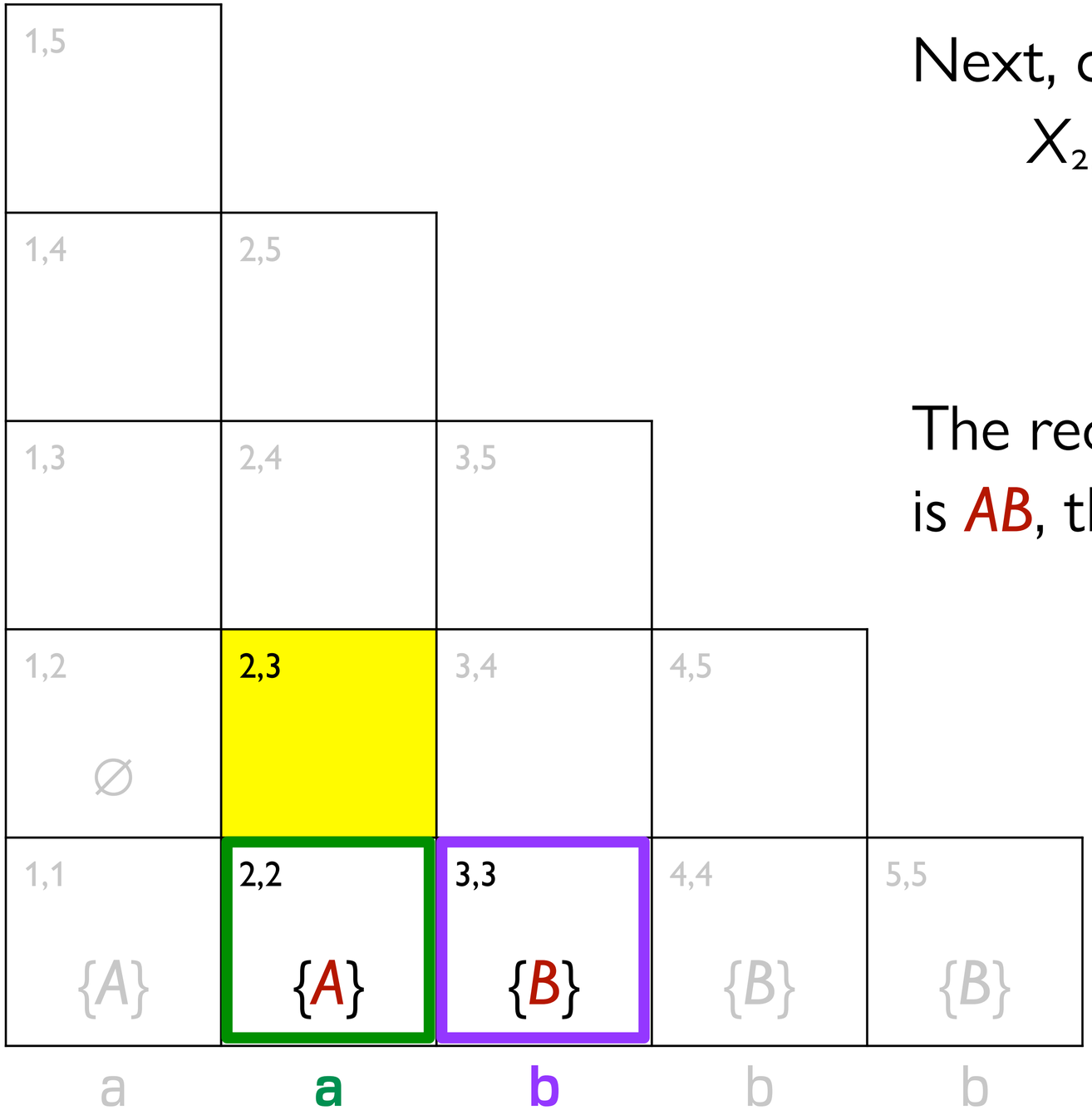
$Q \in X_{2,2},$

$R \in X_{3,3}\}.$

$S \to AB$

$A \to BB \mid a$

$B \to AB \mid b$



Next, compute

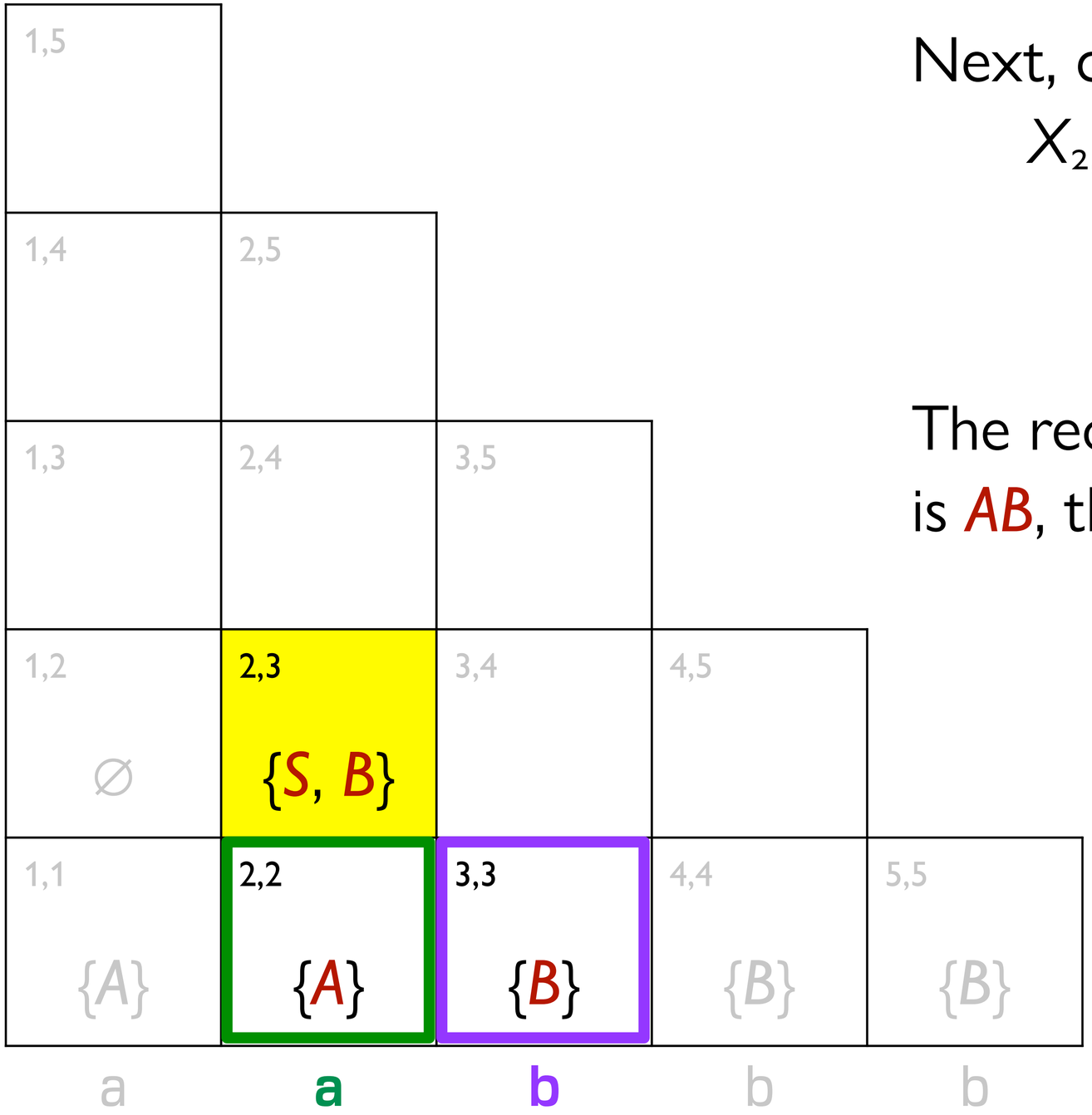
$X_{2,3} = \{P \mid P \to QR, Q \in X_{2,2}, R \in X_{3,3}\}$.

The required right side is AB, thus $X_{2,3} = \{S, B\}$.

$S \to AB$

$A \to BB \mid a$

$B \to AB \mid b$



Next, compute

$X_{2,3} = \{P \mid P \to QR,$

$Q \in X_{2,2},$

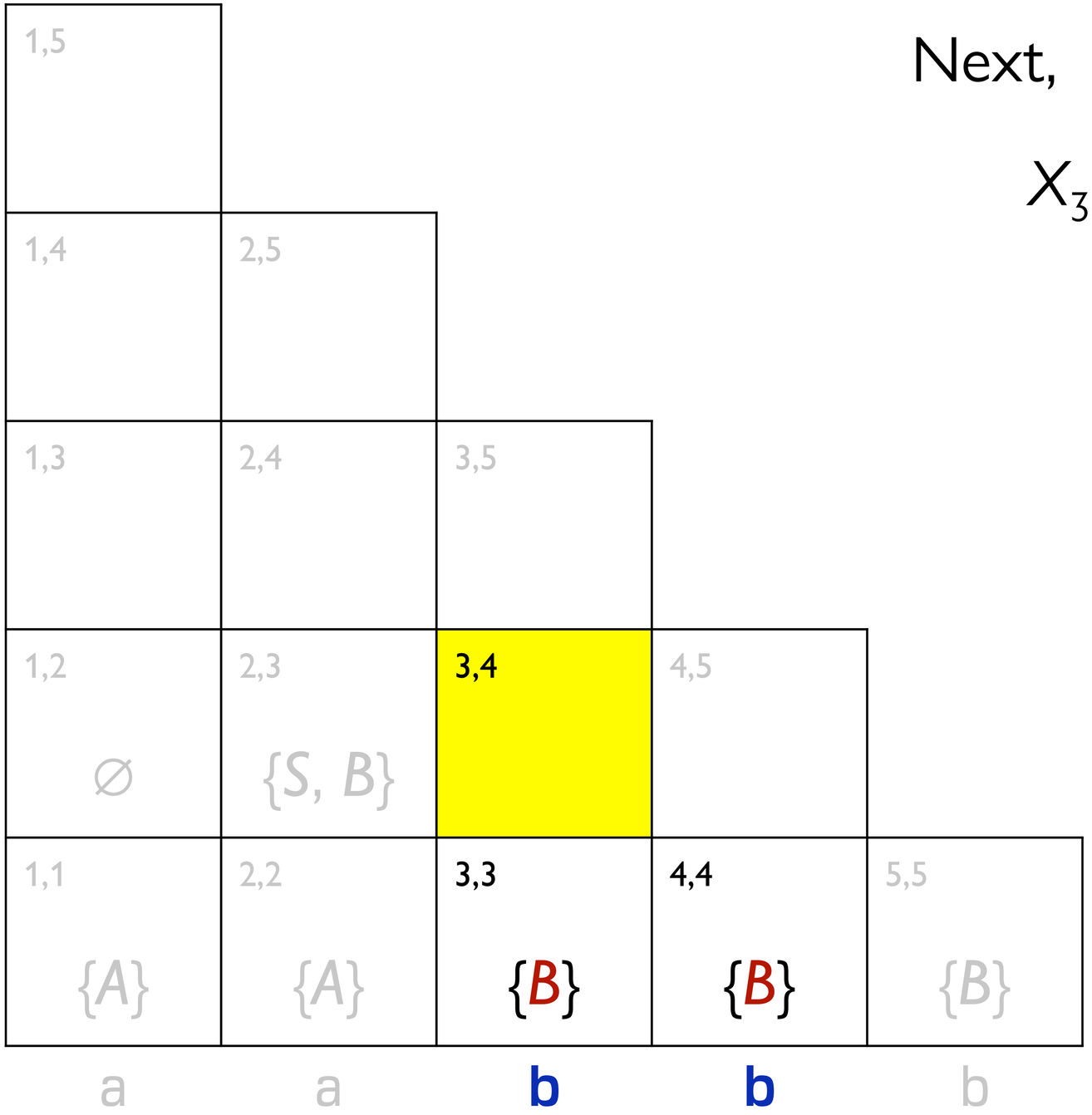
$R \in X_{3,3}\}.$

The required right side is AB, thus $X_{2,3} = \{S, B\}$.

$S \rightarrow AB$

$A \rightarrow BB \mid a$

$B \rightarrow AB \mid b$



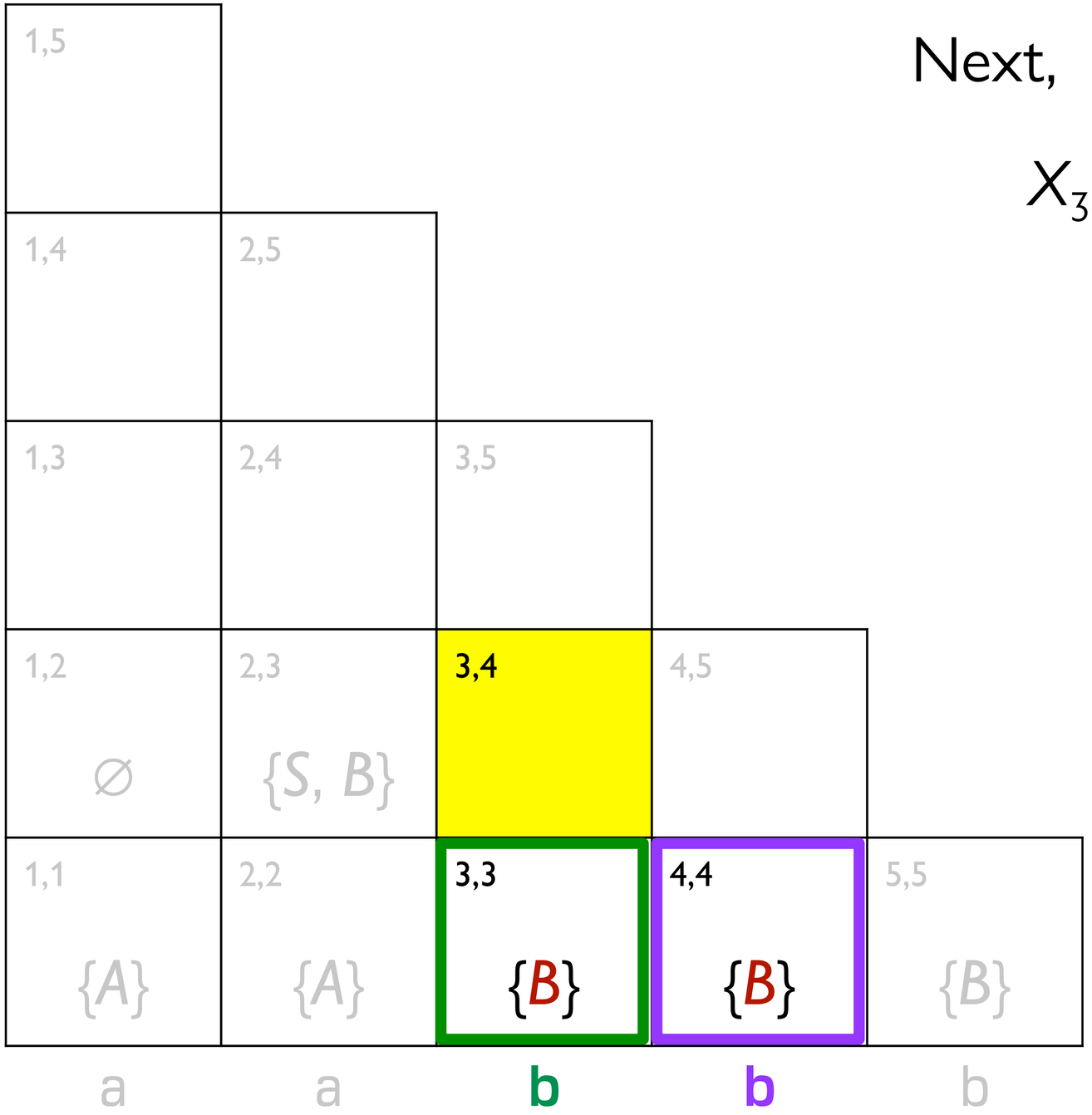
Next,

$X_{3,4} = \ldots$

$S \rightarrow AB$

$A \rightarrow BB \mid a$

$B \rightarrow AB \mid b$



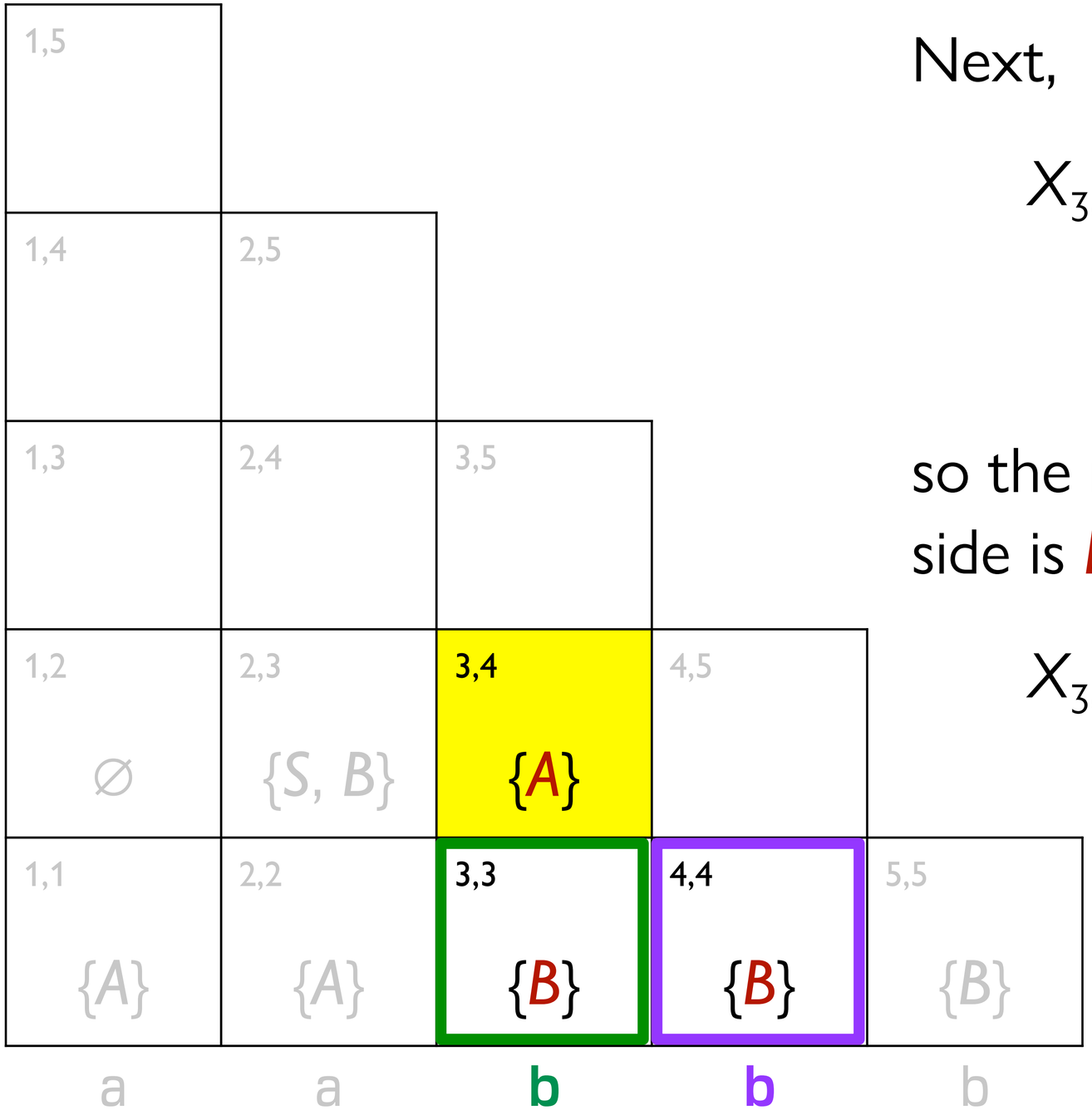
Next,

$$X_{3,4} = \{P \mid P \rightarrow QR, Q \in X_{3,3}, R \in X_{4,4}\}$$

$S \to AB$

$A \to BB \mid a$

$B \to AB \mid b$



Next,

$$X_{3,4} = \{P \mid P \to QR, Q \in X_{3,3}, R \in X_{4,4}\},$$

so the required right side is BB, thus

$$X_{3,4} = \{A\}.$$

$S \rightarrow AB$

$A \rightarrow BB \mid a$

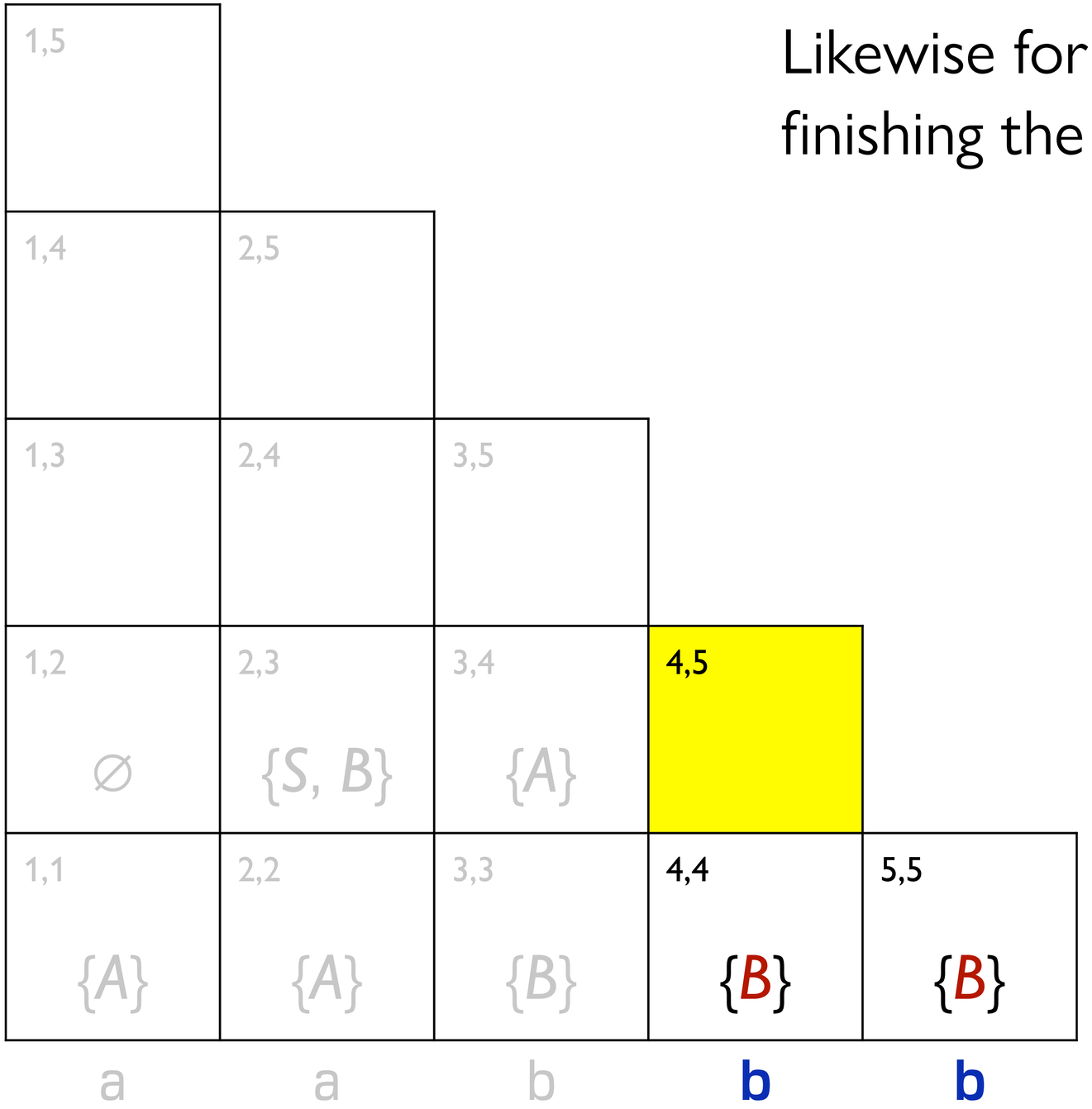
$B \rightarrow AB \mid b$

1,5				
1,4	2,5			
1,3	2,4	3,5		
1,2 $\varnothing$	2,3 $\{S, B\}$	3,4 $\{A\}$	4,5	
1,1 $\{A\}$	2,2 $\{A\}$	3,3 $\{B\}$	4,4 $\{B\}$	5,5 $\{B\}$
a	a	b	b	b

$S \rightarrow AB$

$A \rightarrow BB \mid a$

$B \rightarrow AB \mid b$

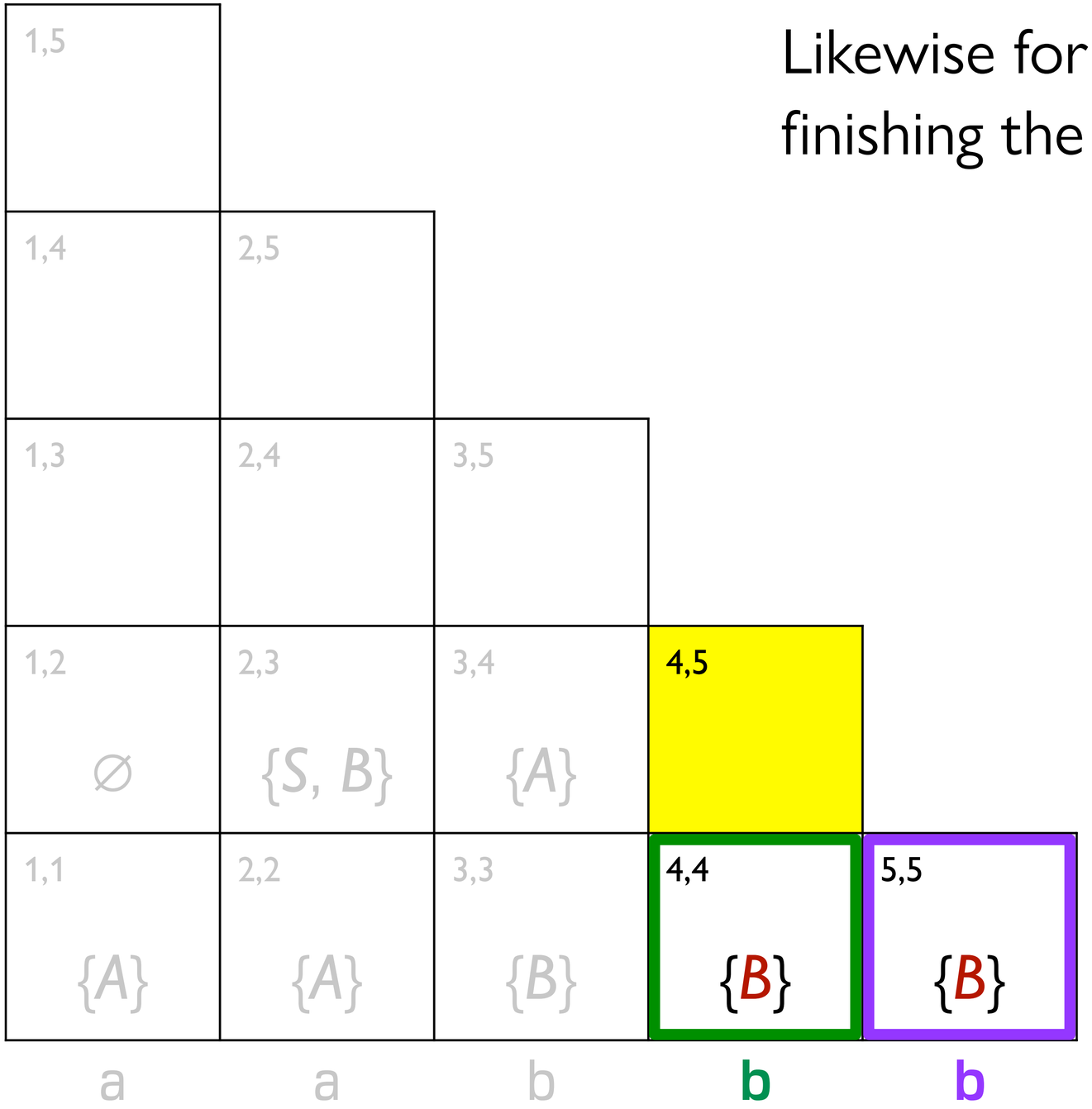


Likewise for $X_{4,5}$, finishing the second row.

$S \rightarrow AB$

$A \rightarrow BB \mid a$

$B \rightarrow AB \mid b$

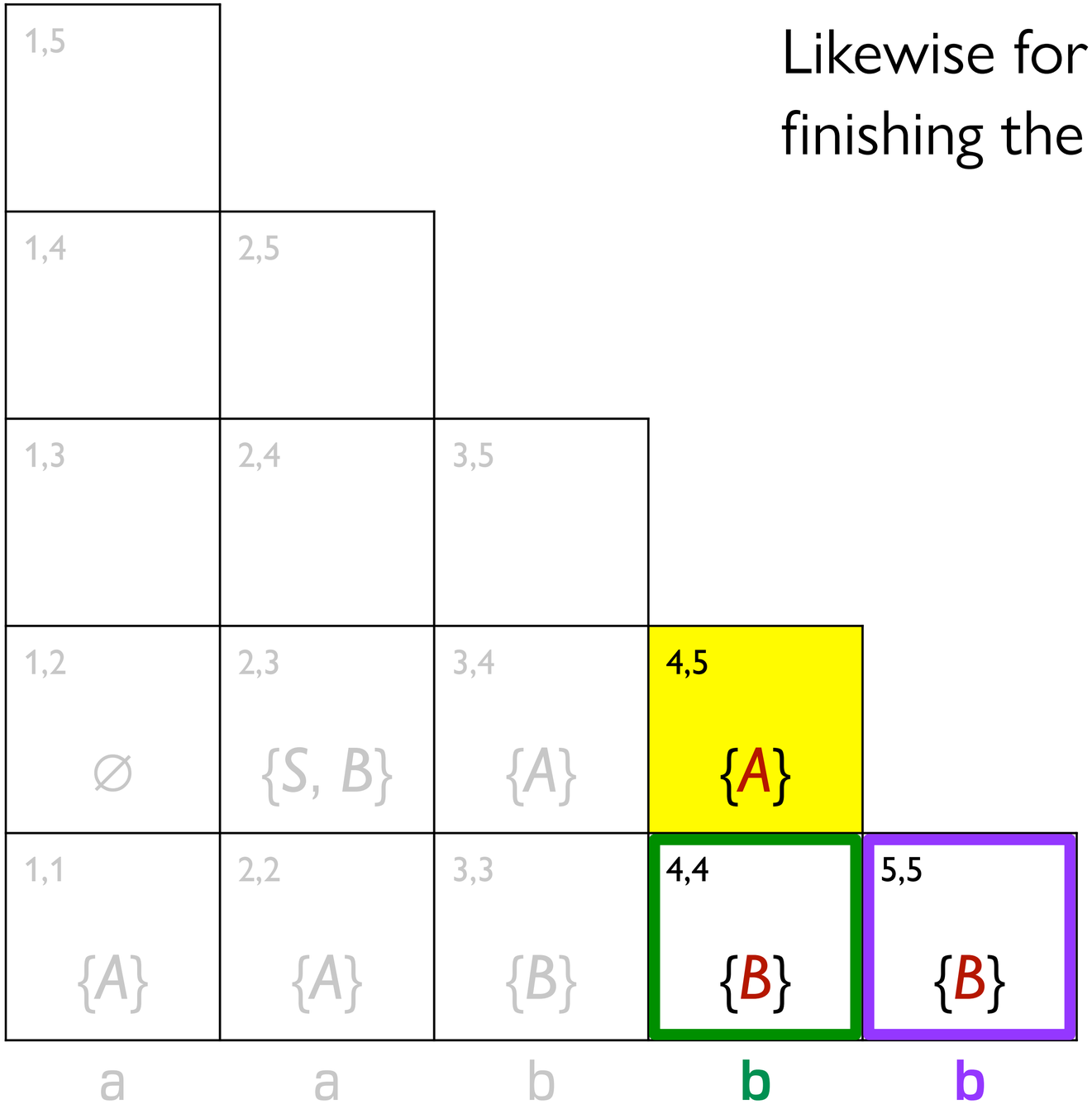


Likewise for $X_{4,5}$,
finishing the second row.

$S \rightarrow AB$

$A \rightarrow BB \mid a$

$B \rightarrow AB \mid b$



Likewise for $X_{4,5}$,
finishing the second row.

$S \rightarrow AB$

$A \rightarrow BB \mid a$

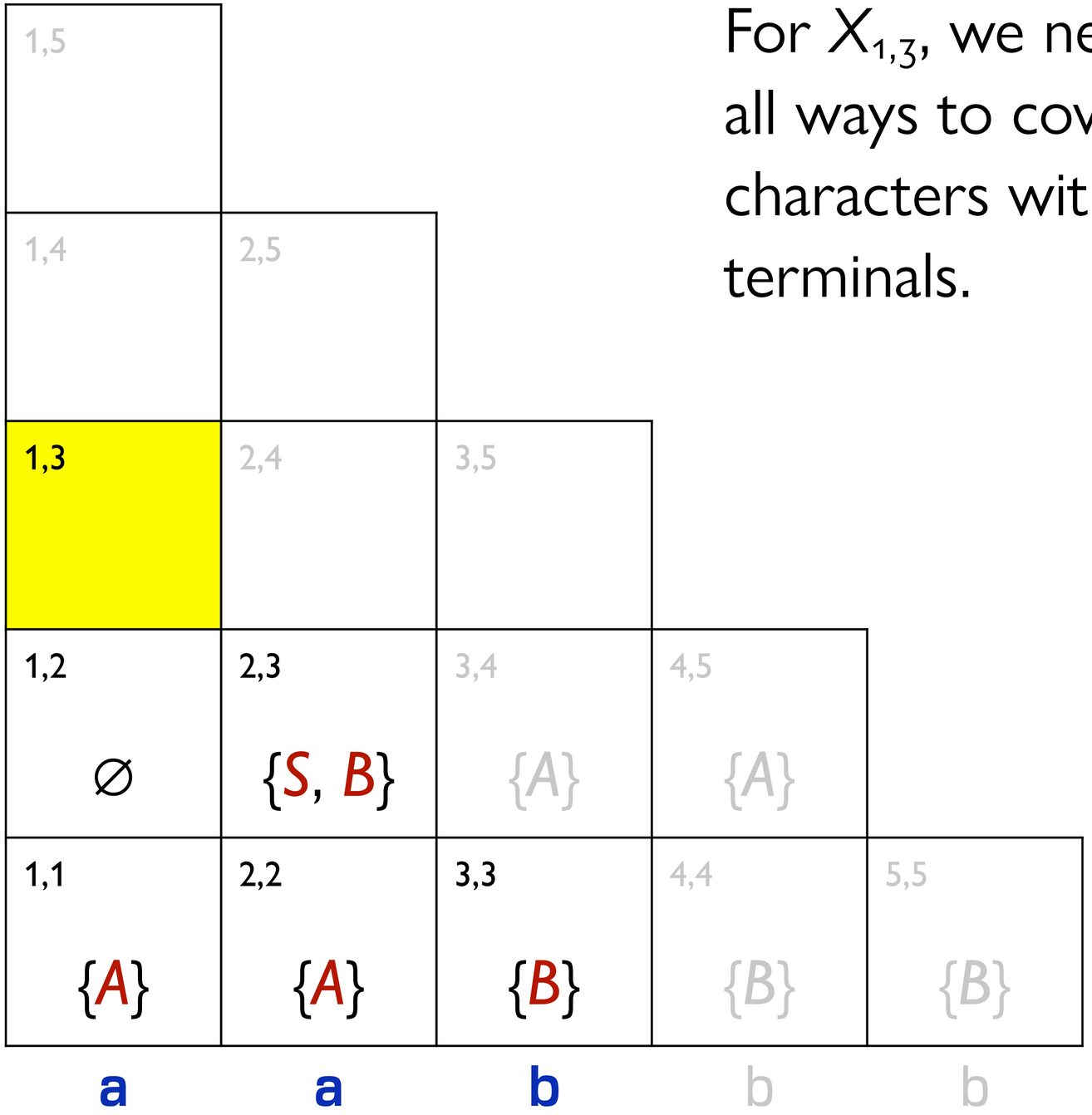
$B \rightarrow AB \mid b$

1,5				
1,4	2,5			
1,3	2,4	3,5		
1,2 $\varnothing$	2,3 $\{S, B\}$	3,4 $\{A\}$	4,5 $\{A\}$	
1,1 $\{A\}$	2,2 $\{A\}$	3,3 $\{B\}$	4,4 $\{B\}$	5,5 $\{B\}$
a	a	b	b	b

$S \rightarrow AB$

$A \rightarrow BB \mid a$

$B \rightarrow AB \mid b$

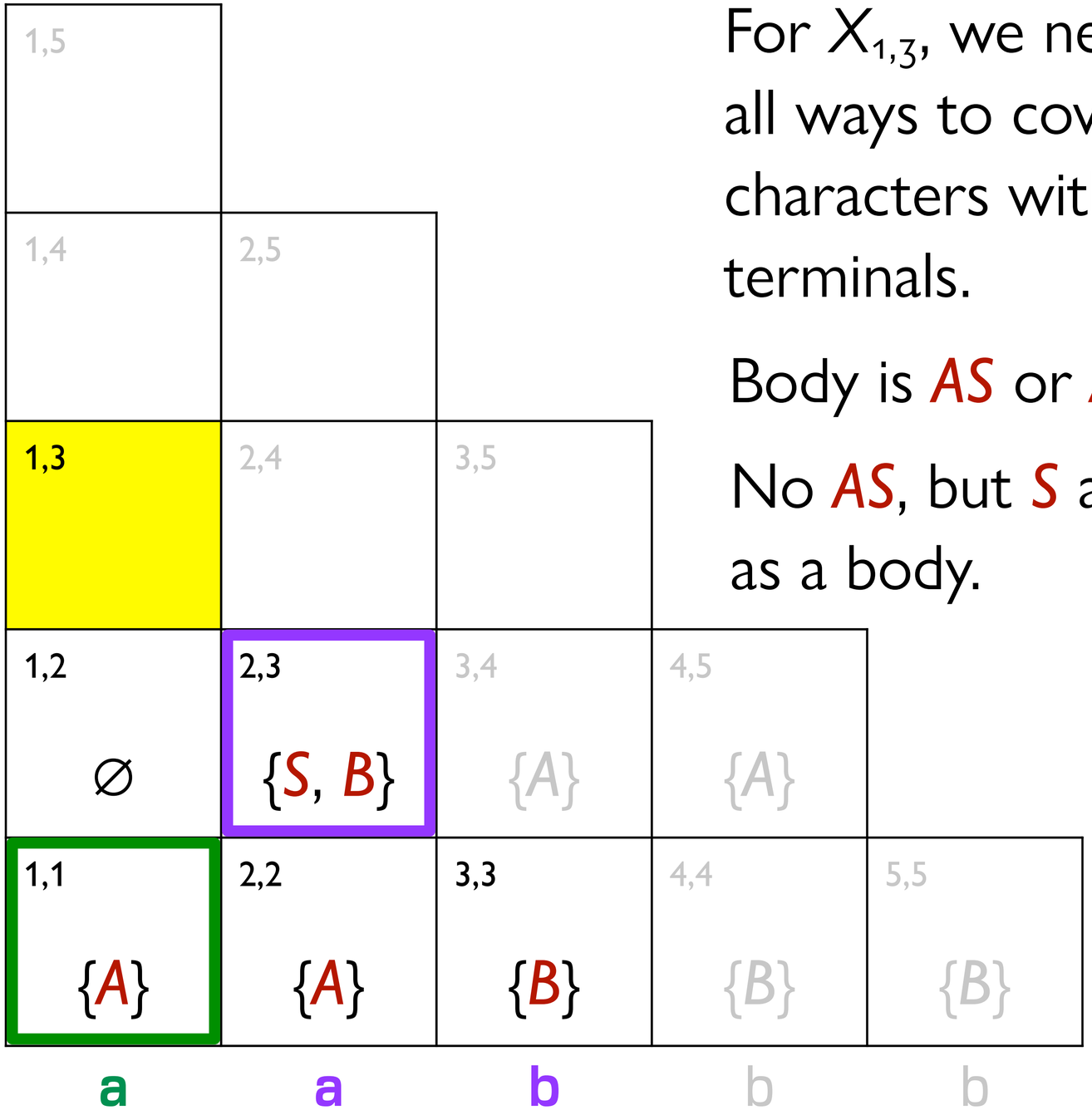


For $X_{1,3}$, we need to consider all ways to cover those three characters with two non-terminals.

$S \rightarrow AB$

$A \rightarrow BB \mid a$

$B \rightarrow AB \mid b$



For $X_{1,3}$, we need to consider all ways to cover those three characters with two non-terminals.

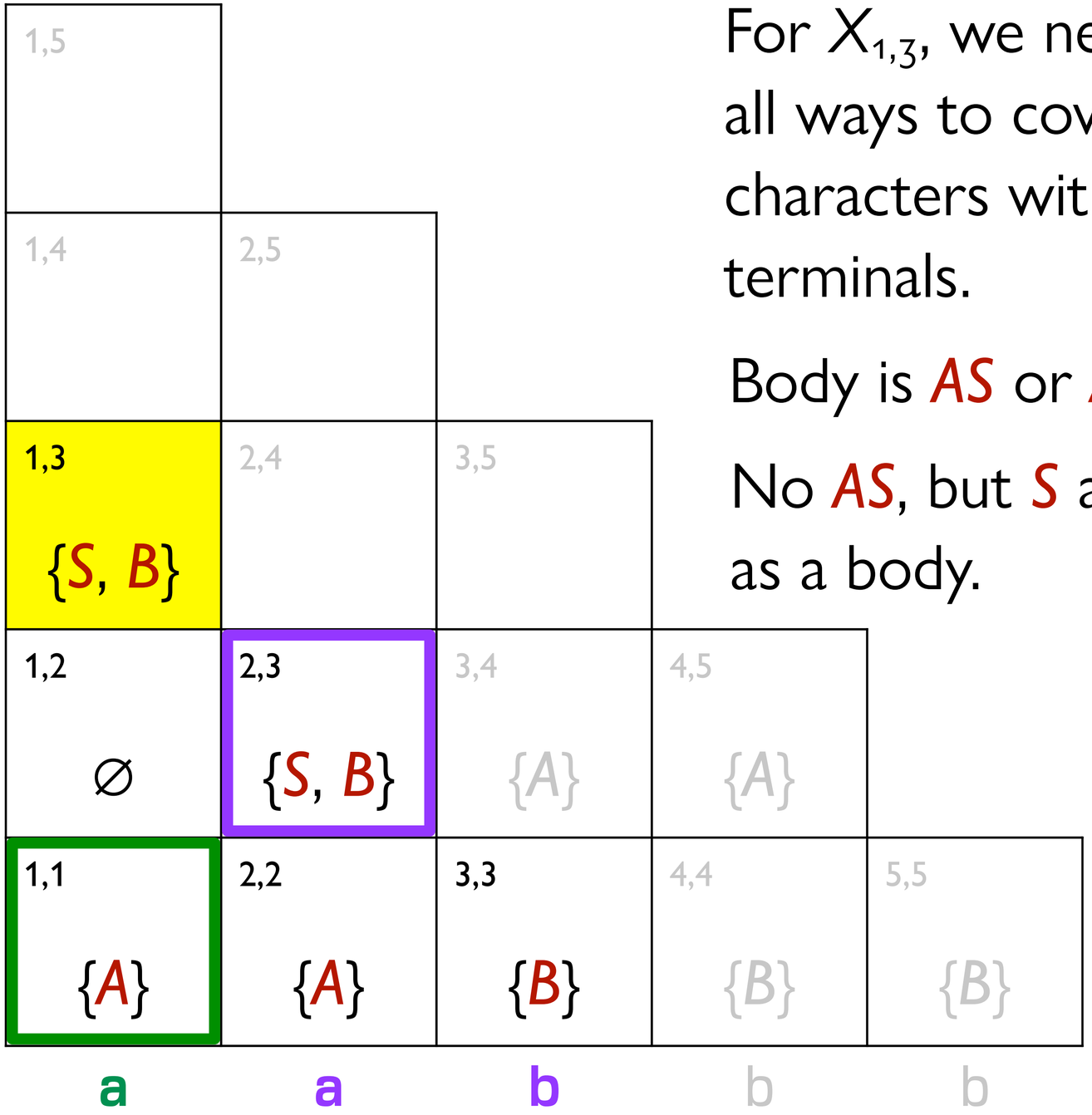
Body is AS or AB.

No AS, but S and B have AB as a body.

$S \rightarrow AB$

$A \rightarrow BB \mid a$

$B \rightarrow AB \mid b$



For $X_{1,3}$, we need to consider all ways to cover those three characters with two non-terminals.

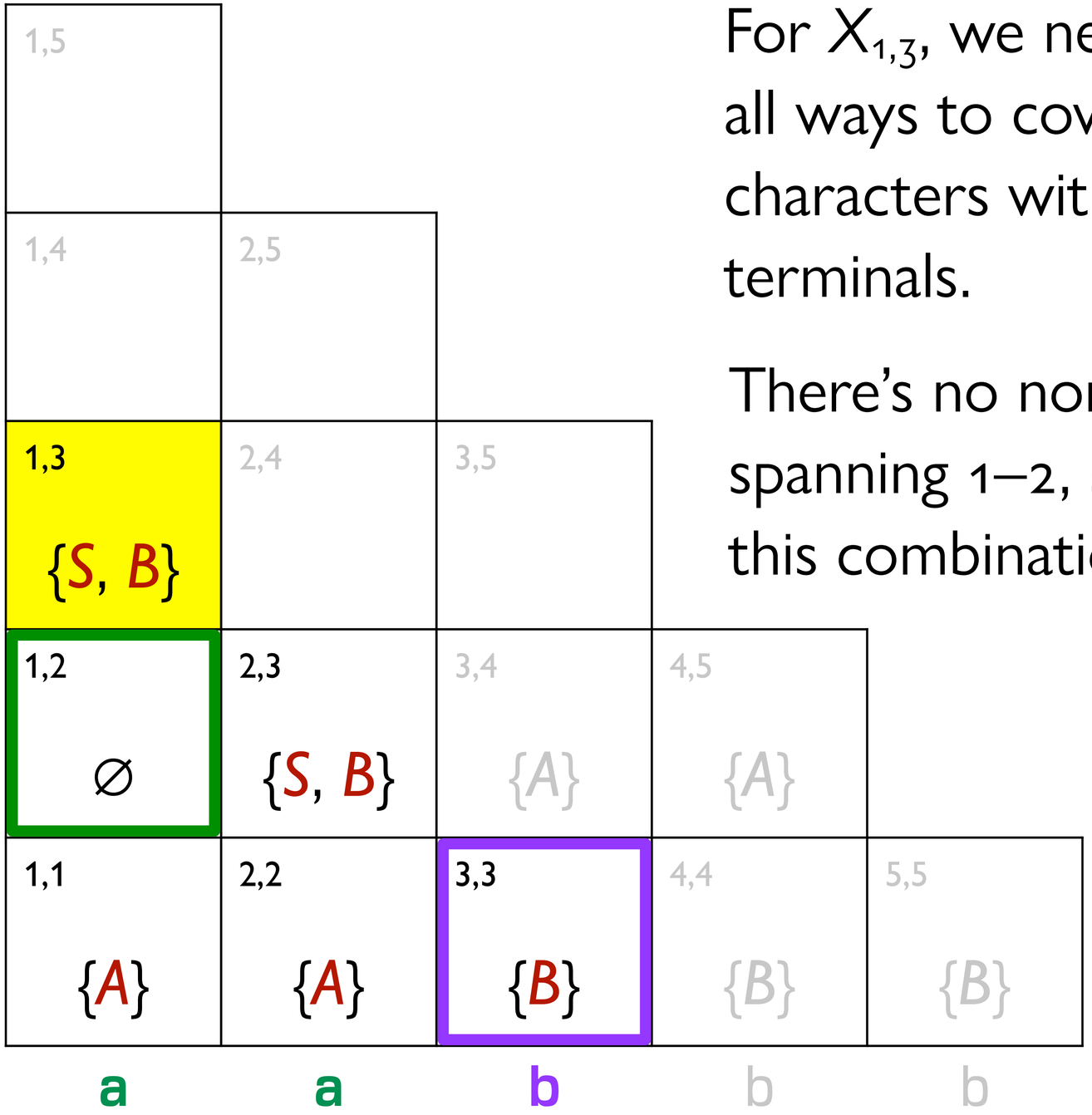
Body is AS or AB.

No AS, but S and B have AB as a body.

$S \rightarrow AB$

$A \rightarrow BB \mid a$

$B \rightarrow AB \mid b$



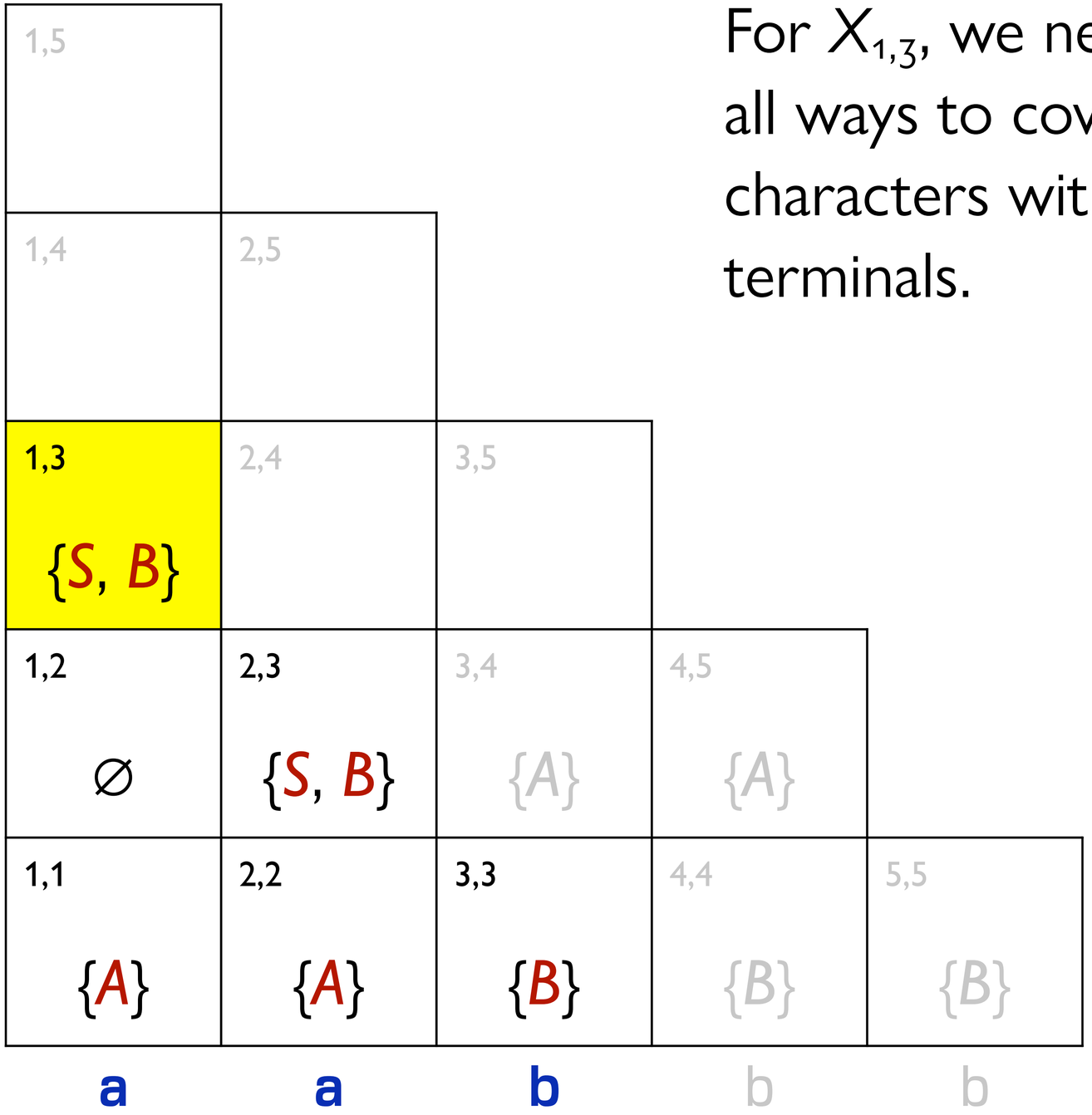
For $X_{1,3}$, we need to consider all ways to cover those three characters with two non-terminals.

There's no non-terminal spanning 1–2, so we can't use this combination.

$S \rightarrow AB$

$A \rightarrow BB \mid a$

$B \rightarrow AB \mid b$



For $X_{1,3}$, we need to consider all ways to cover those three characters with two non-terminals.

$S \rightarrow AB$

$A \rightarrow BB \mid a$

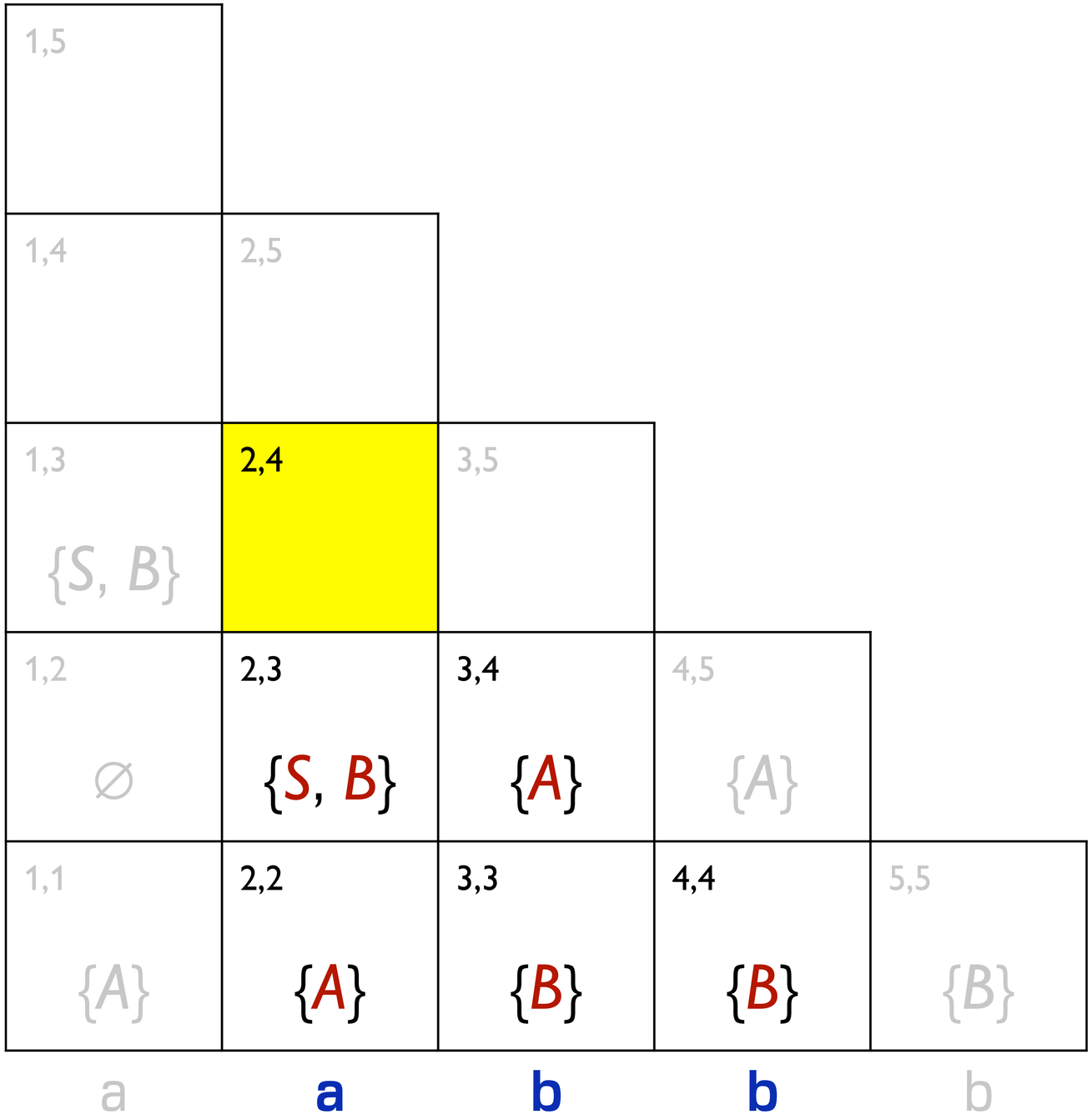
$B \rightarrow AB \mid b$

1,5				
1,4	2,5			
1,3 $\{S, B\}$	2,4	3,5		
1,2 $\varnothing$	2,3 $\{S, B\}$	3,4 $\{A\}$	4,5 $\{A\}$	
1,1 $\{A\}$	2,2 $\{A\}$	3,3 $\{B\}$	4,4 $\{B\}$	5,5 $\{B\}$
a	a	b	b	b

$S \rightarrow AB$

$A \rightarrow BB \mid a$

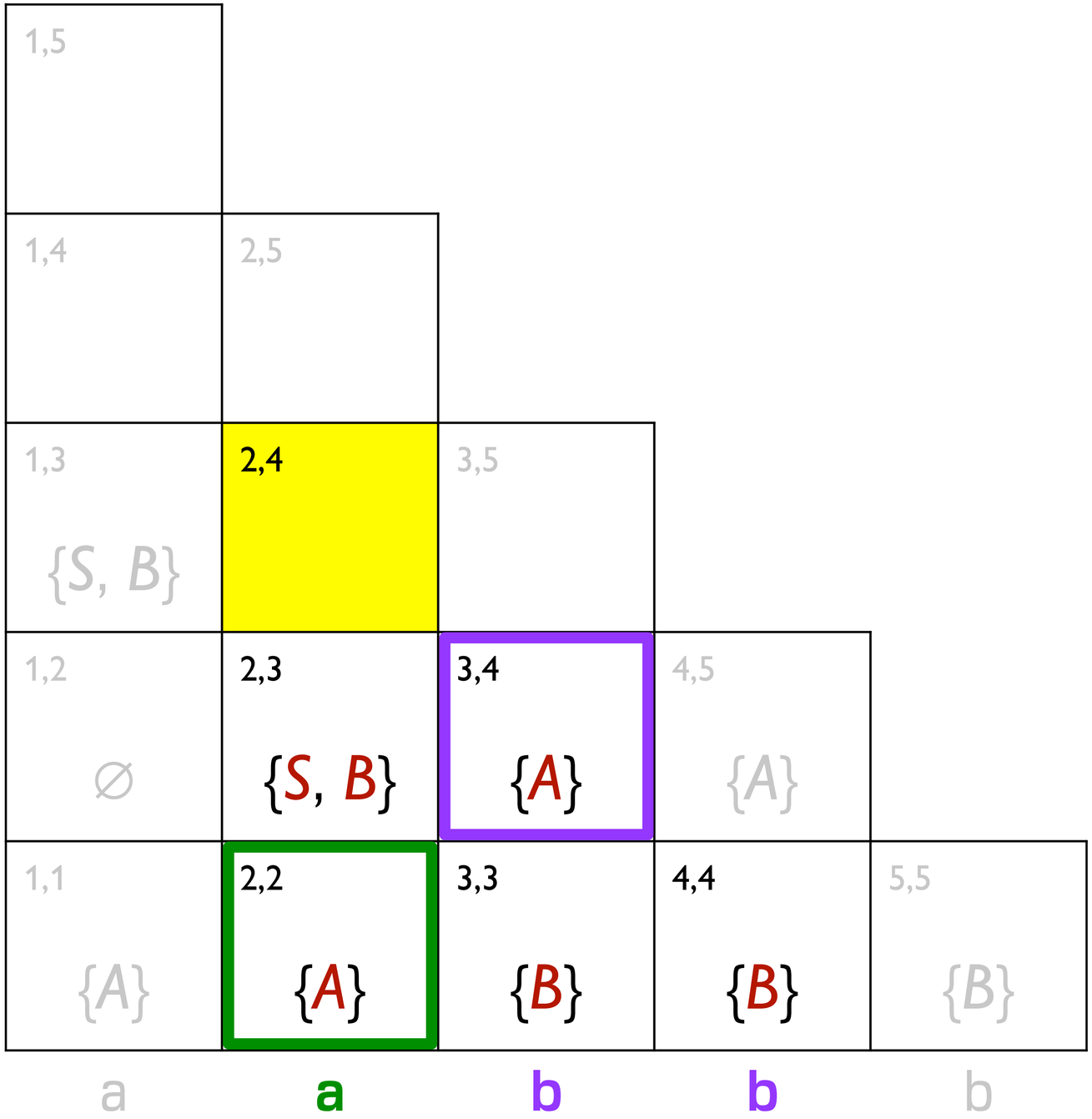
$B \rightarrow AB \mid b$



$S \rightarrow AB$

$A \rightarrow BB \mid a$

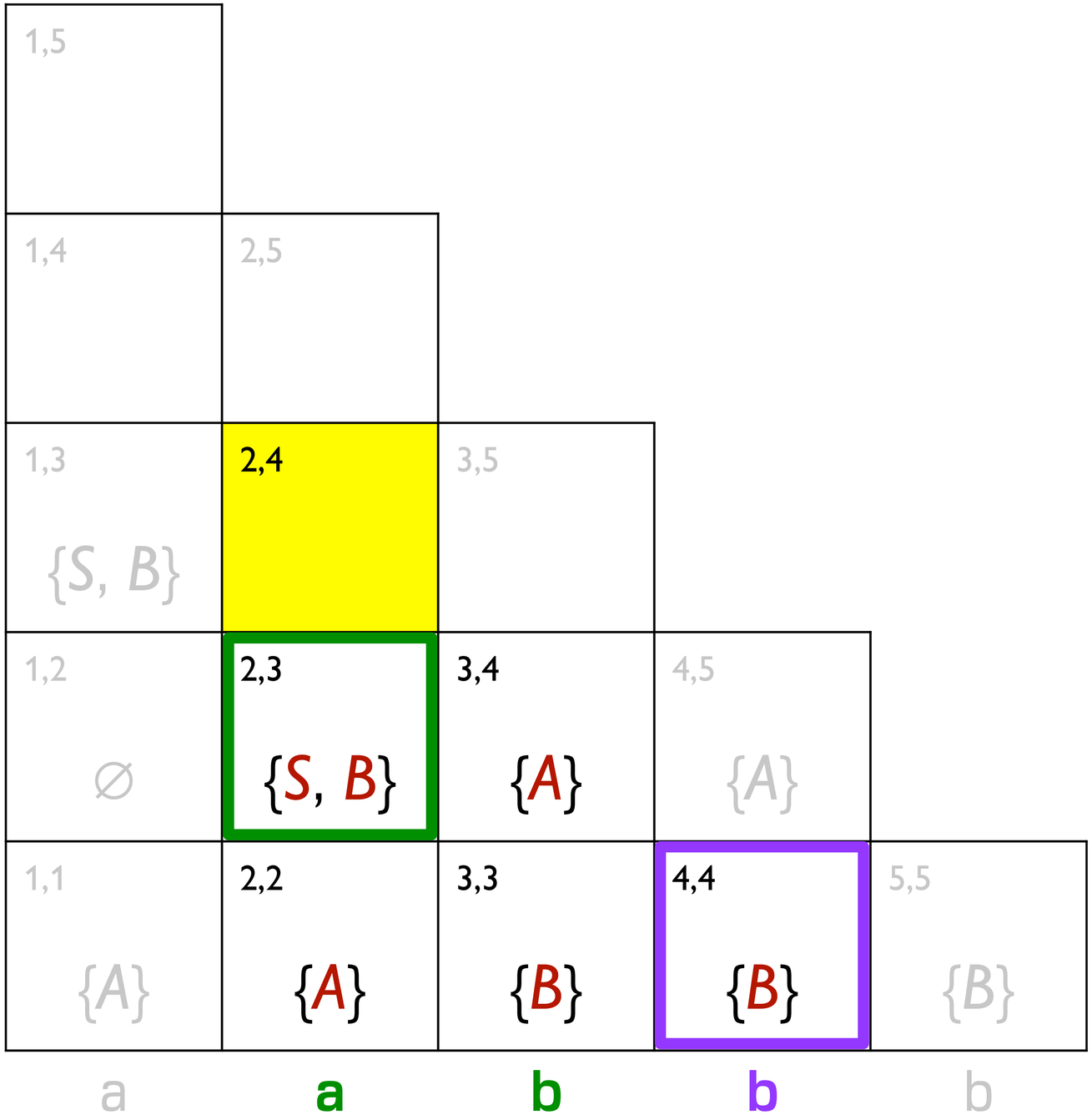
$B \rightarrow AB \mid b$



$S \rightarrow AB$

$A \rightarrow BB \mid a$

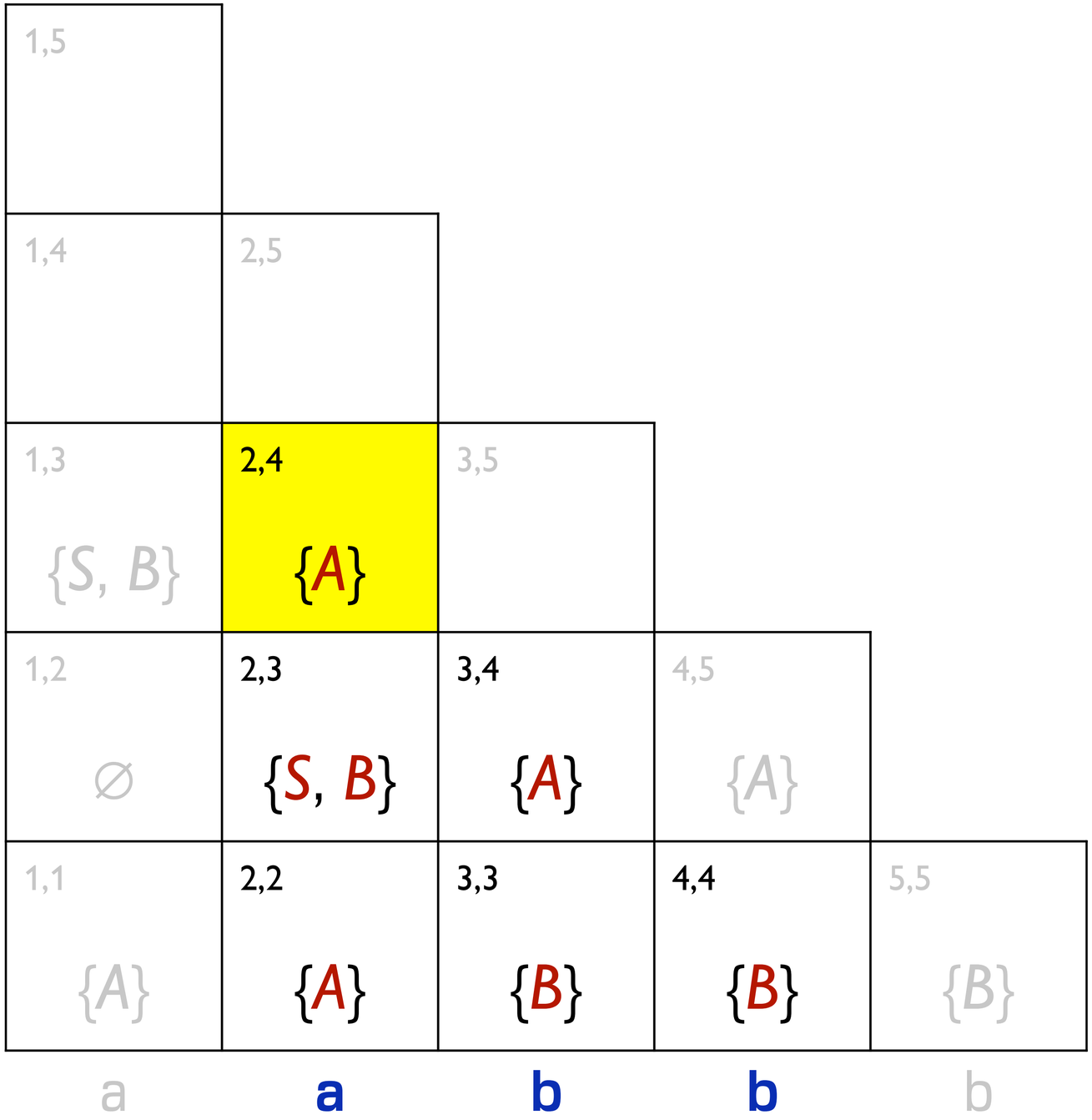
$B \rightarrow AB \mid b$



$S \rightarrow AB$

$A \rightarrow BB \mid a$

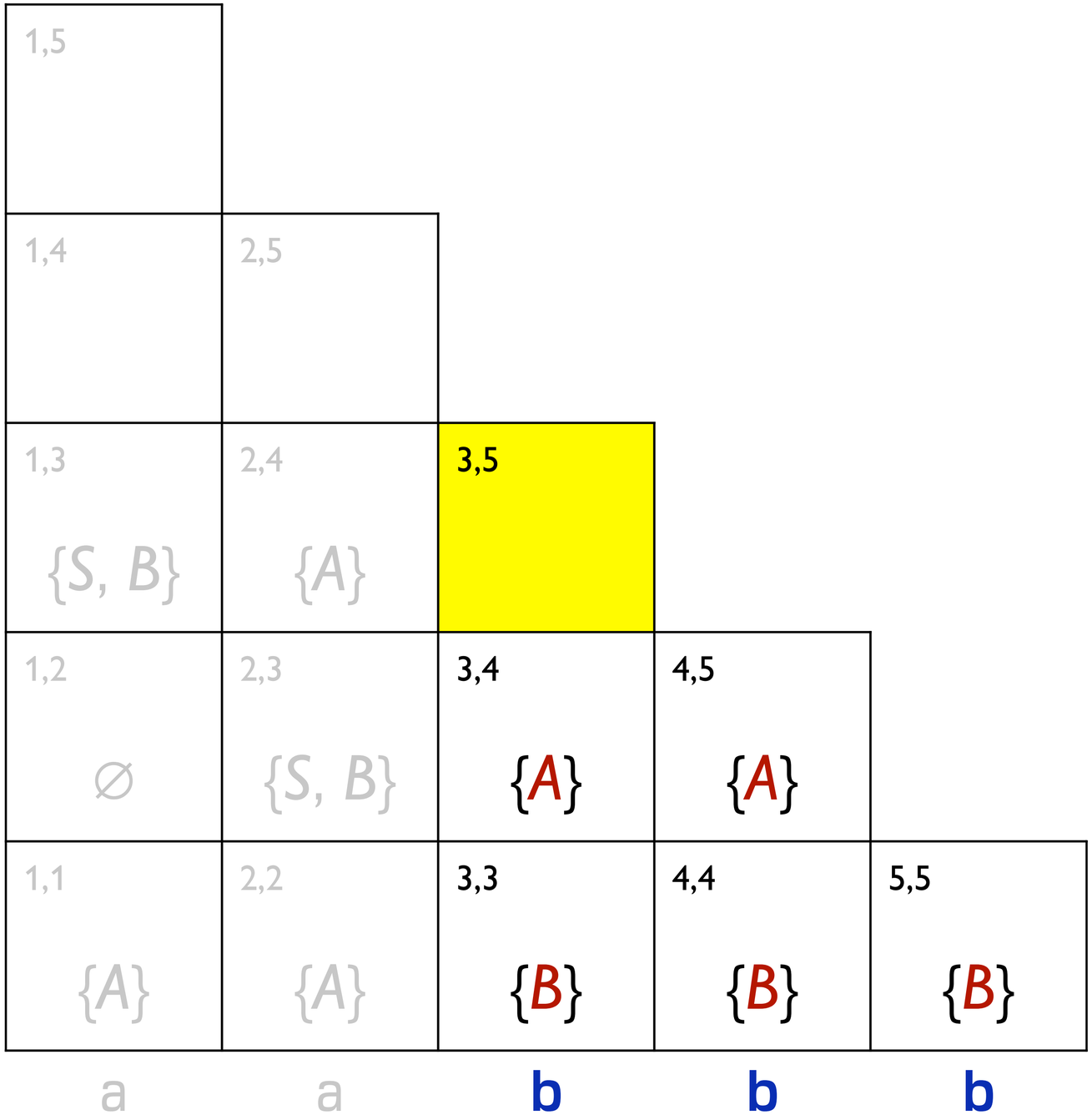
$B \rightarrow AB \mid b$



$S \rightarrow AB$

$A \rightarrow BB \mid a$

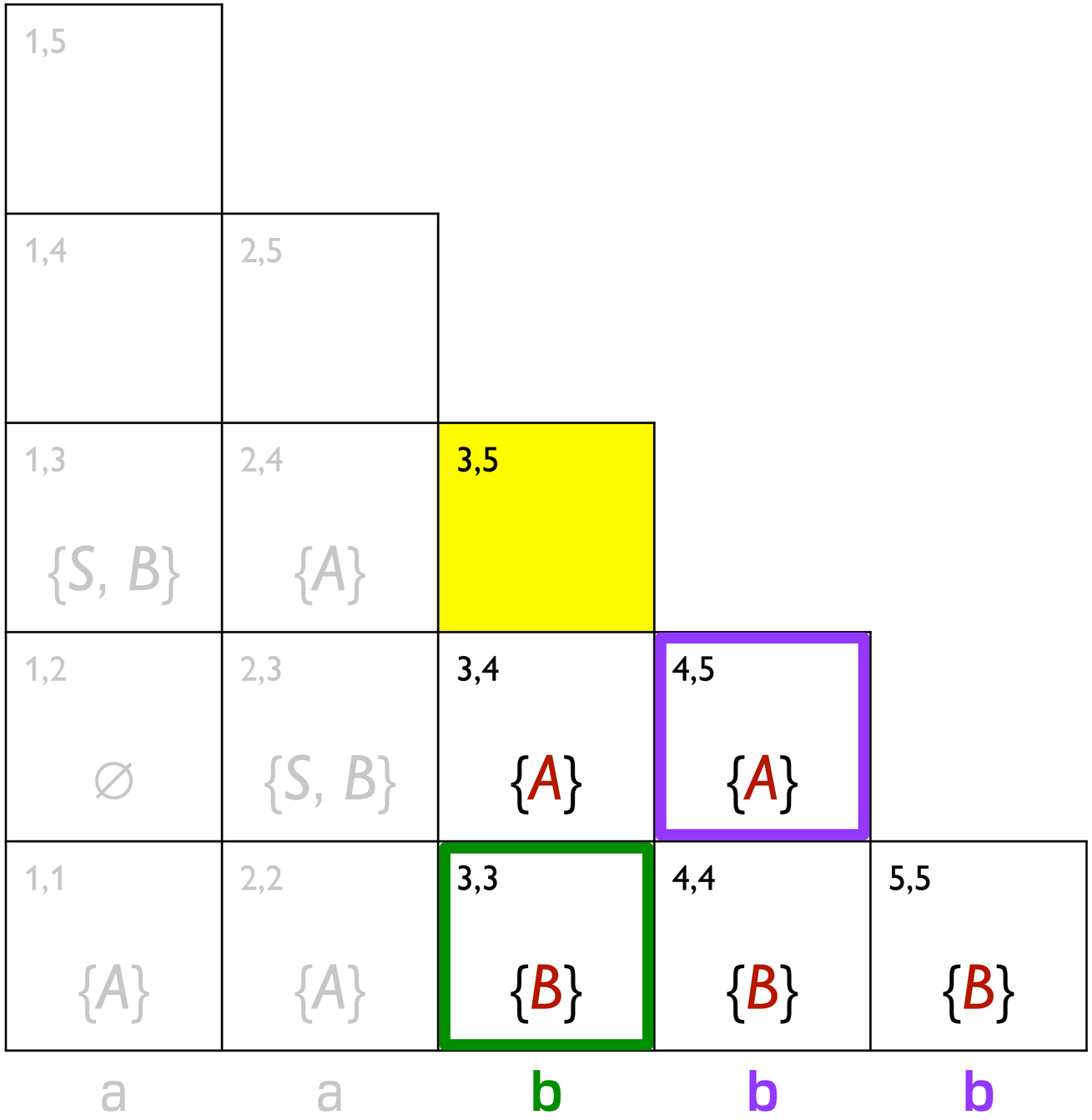
$B \rightarrow AB \mid b$



$S \rightarrow AB$

$A \rightarrow BB \mid a$

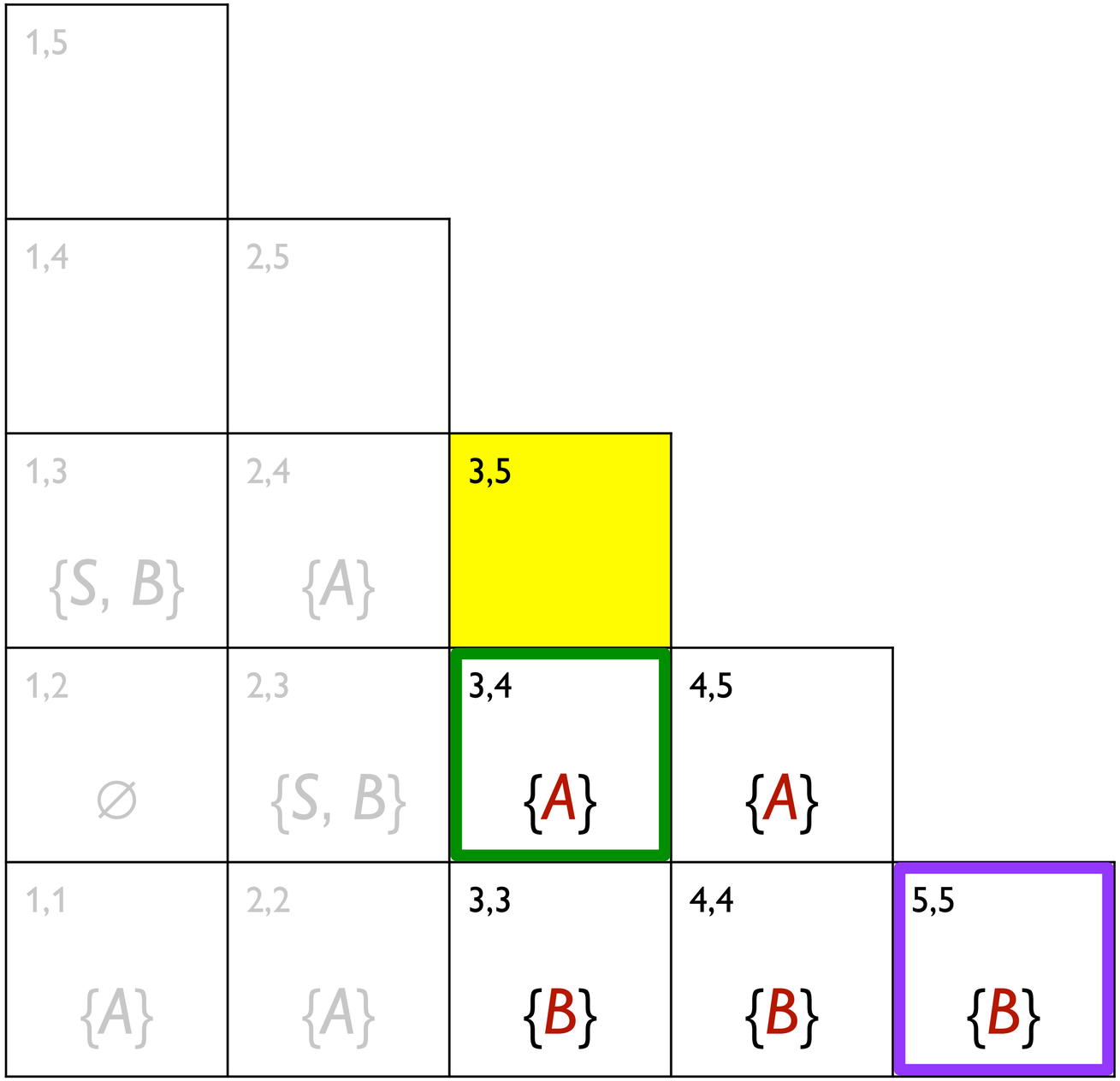
$B \rightarrow AB \mid b$



$S \rightarrow AB$

$A \rightarrow BB \mid a$

$B \rightarrow AB \mid b$



$S \rightarrow AB$

$A \rightarrow BB \mid a$

$B \rightarrow AB \mid b$

1,5				
1,4	2,5			
1,3 $\{S, B\}$	2,4 $\{A\}$	3,5 $\{S, B\}$		
1,2 $\varnothing$	2,3 $\{S, B\}$	3,4 $\{A\}$	4,5 $\{A\}$	
1,1 $\{A\}$	2,2 $\{A\}$	3,3 $\{B\}$	4,4 $\{B\}$	5,5 $\{B\}$
a	a	**b**	**b**	**b**

$S \rightarrow AB$

$A \rightarrow BB \mid a$

$B \rightarrow AB \mid b$

1,5				
1,4	2,5			
1,3 $\{S, B\}$	2,4 $\{A\}$	3,5 $\{S, B\}$		
1,2 $\varnothing$	2,3 $\{S, B\}$	3,4 $\{A\}$	4,5 $\{A\}$	
1,1 $\{A\}$	2,2 $\{A\}$	3,3 $\{B\}$	4,4 $\{B\}$	5,5 $\{B\}$
a	a	b	b	b

$S \rightarrow AB$

$A \rightarrow BB \mid a$

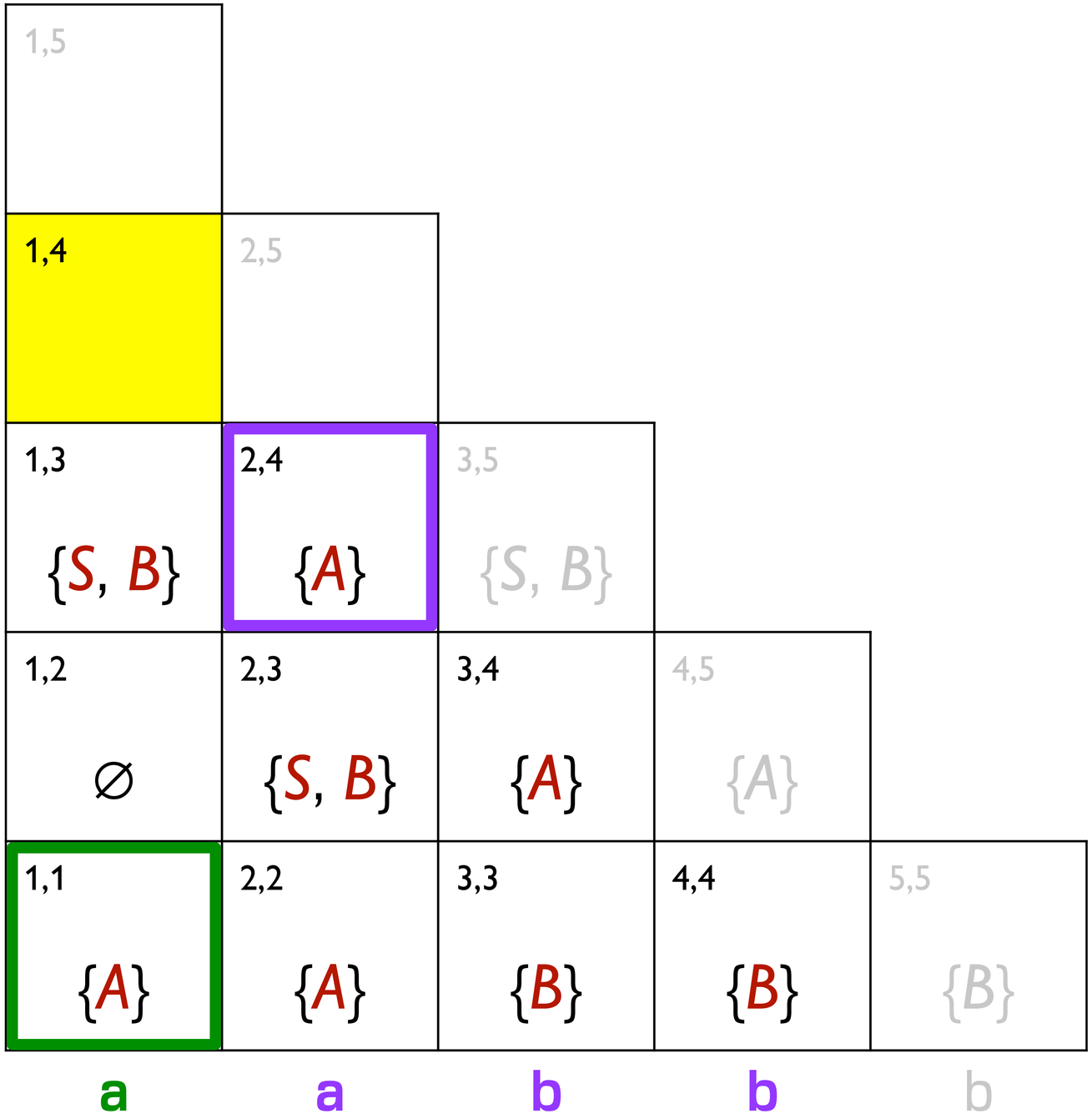
$B \rightarrow AB \mid b$

1,5				
1,4	2,5			
1,3 $\{S, B\}$	2,4 $\{A\}$	3,5 $\{S, B\}$		
1,2 $\varnothing$	2,3 $\{S, B\}$	3,4 $\{A\}$	4,5 $\{A\}$	
1,1 $\{A\}$	2,2 $\{A\}$	3,3 $\{B\}$	4,4 $\{B\}$	5,5 $\{B\}$
a	**a**	**b**	**b**	b

$S \rightarrow AB$

$A \rightarrow BB \mid a$

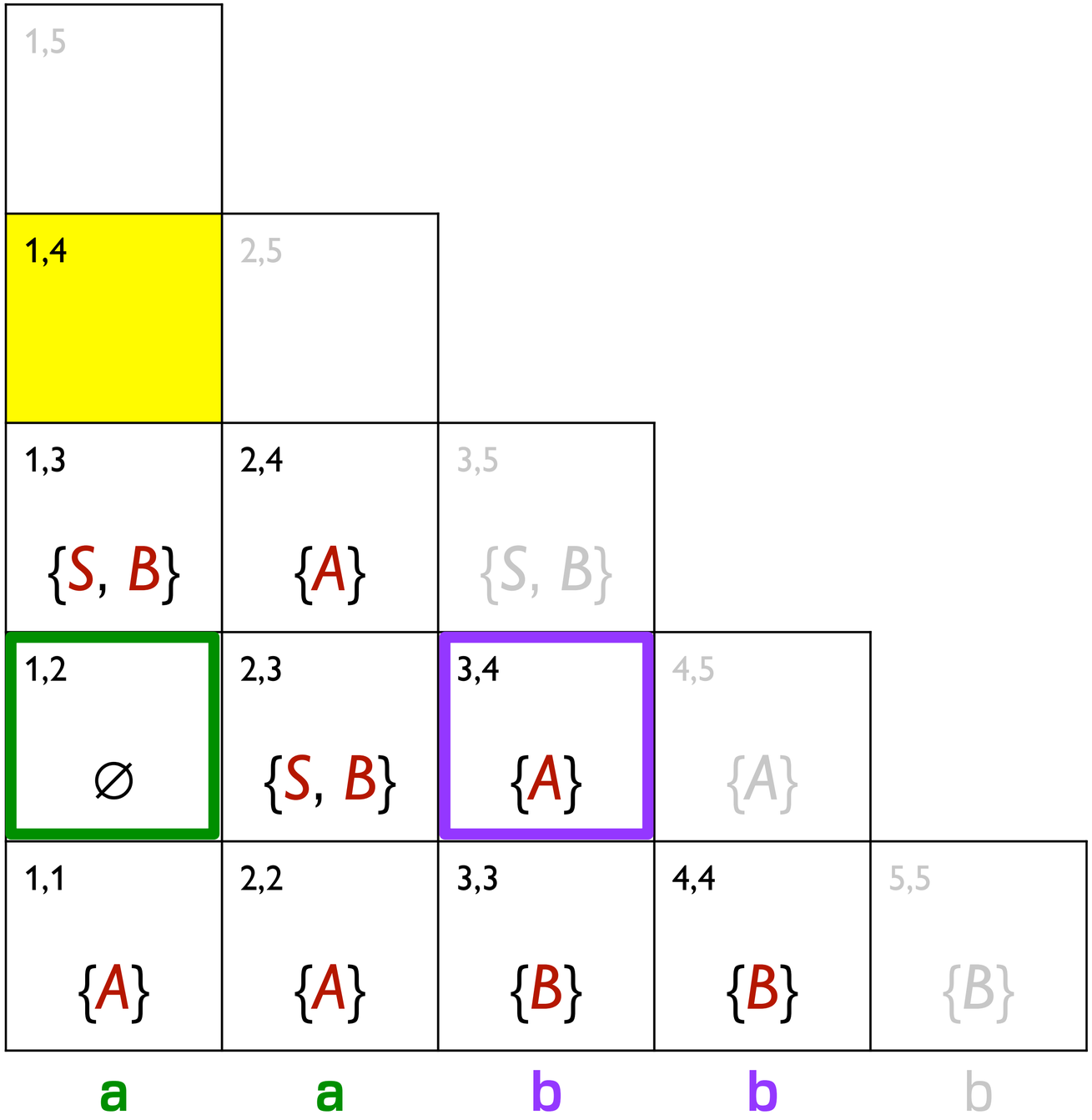
$B \rightarrow AB \mid b$



$S \rightarrow AB$

$A \rightarrow BB \mid a$

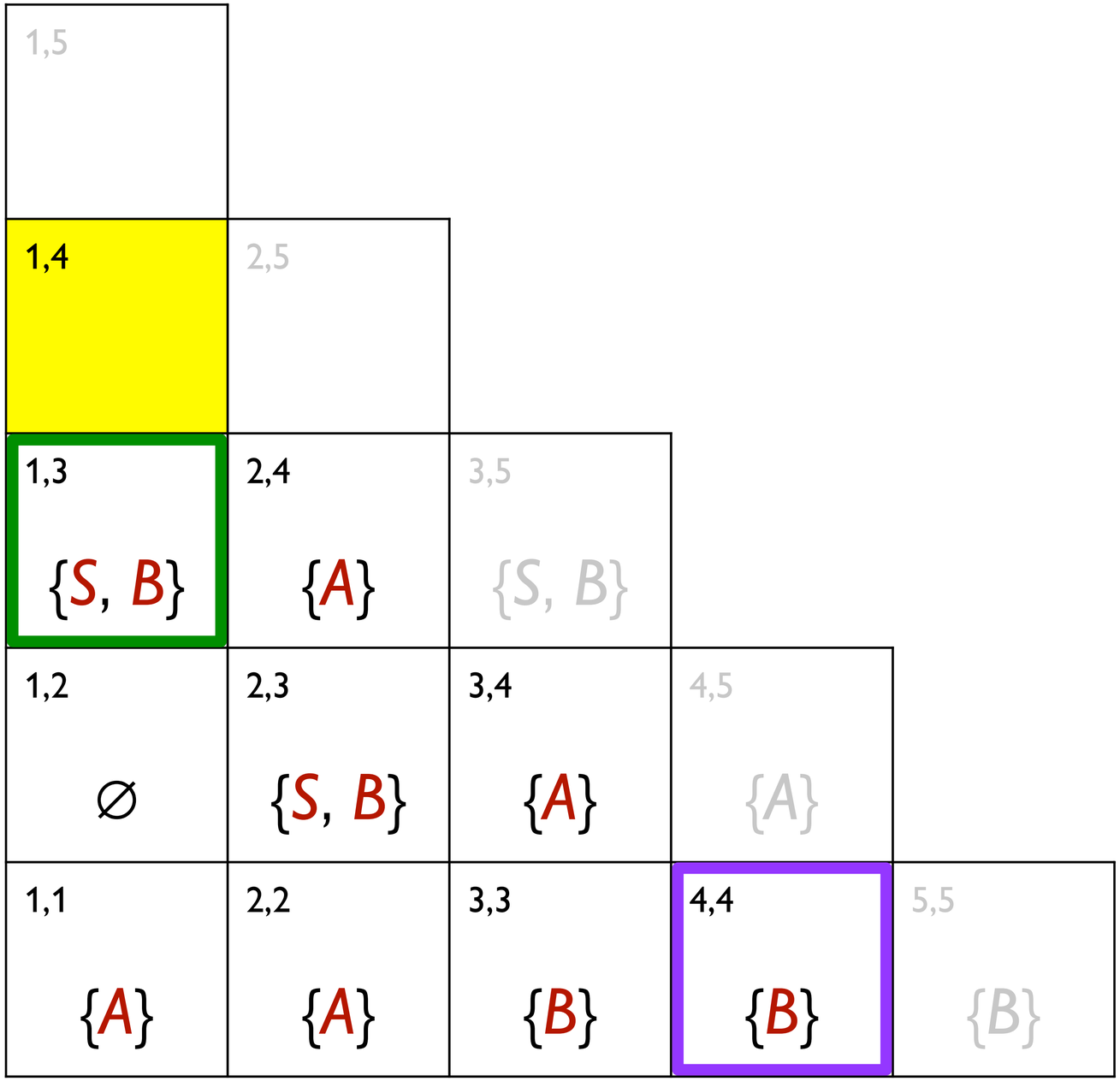
$B \rightarrow AB \mid b$



$S \rightarrow AB$

$A \rightarrow BB \mid a$

$B \rightarrow AB \mid b$



$S \rightarrow AB$

$A \rightarrow BB \mid a$

$B \rightarrow AB \mid b$

1,5				
1,4 $\{A\}$	2,5			
1,3 $\{S, B\}$	2,4 $\{A\}$	3,5 $\{S, B\}$		
1,2 $\varnothing$	2,3 $\{S, B\}$	3,4 $\{A\}$	4,5 $\{A\}$	
1,1 $\{A\}$	2,2 $\{A\}$	3,3 $\{B\}$	4,4 $\{B\}$	5,5 $\{B\}$
a	**a**	**b**	**b**	b

$S \rightarrow AB$

$A \rightarrow BB \mid a$

$B \rightarrow AB \mid b$

1,5				
1,4 $\{A\}$	2,5			
1,3 $\{S, B\}$	2,4 $\{A\}$	3,5 $\{S, B\}$		
1,2 $\varnothing$	2,3 $\{S, B\}$	3,4 $\{A\}$	4,5 $\{A\}$	
1,1 $\{A\}$	2,2 $\{A\}$	3,3 $\{B\}$	4,4 $\{B\}$	5,5 $\{B\}$
a	**a**	**b**	**b**	b

$S \rightarrow AB$

$A \rightarrow BB \mid a$

$B \rightarrow AB \mid b$

1,5				
1,4 $\{A\}$	2,5			
1,3 $\{S, B\}$	2,4 $\{A\}$	3,5 $\{S, B\}$		
1,2 $\varnothing$	2,3 $\{S, B\}$	3,4 $\{A\}$	4,5 $\{A\}$	
1,1 $\{A\}$	2,2 $\{A\}$	3,3 $\{B\}$	4,4 $\{B\}$	5,5 $\{B\}$
a	a	b	b	b

$S \rightarrow AB$

$A \rightarrow BB \mid a$

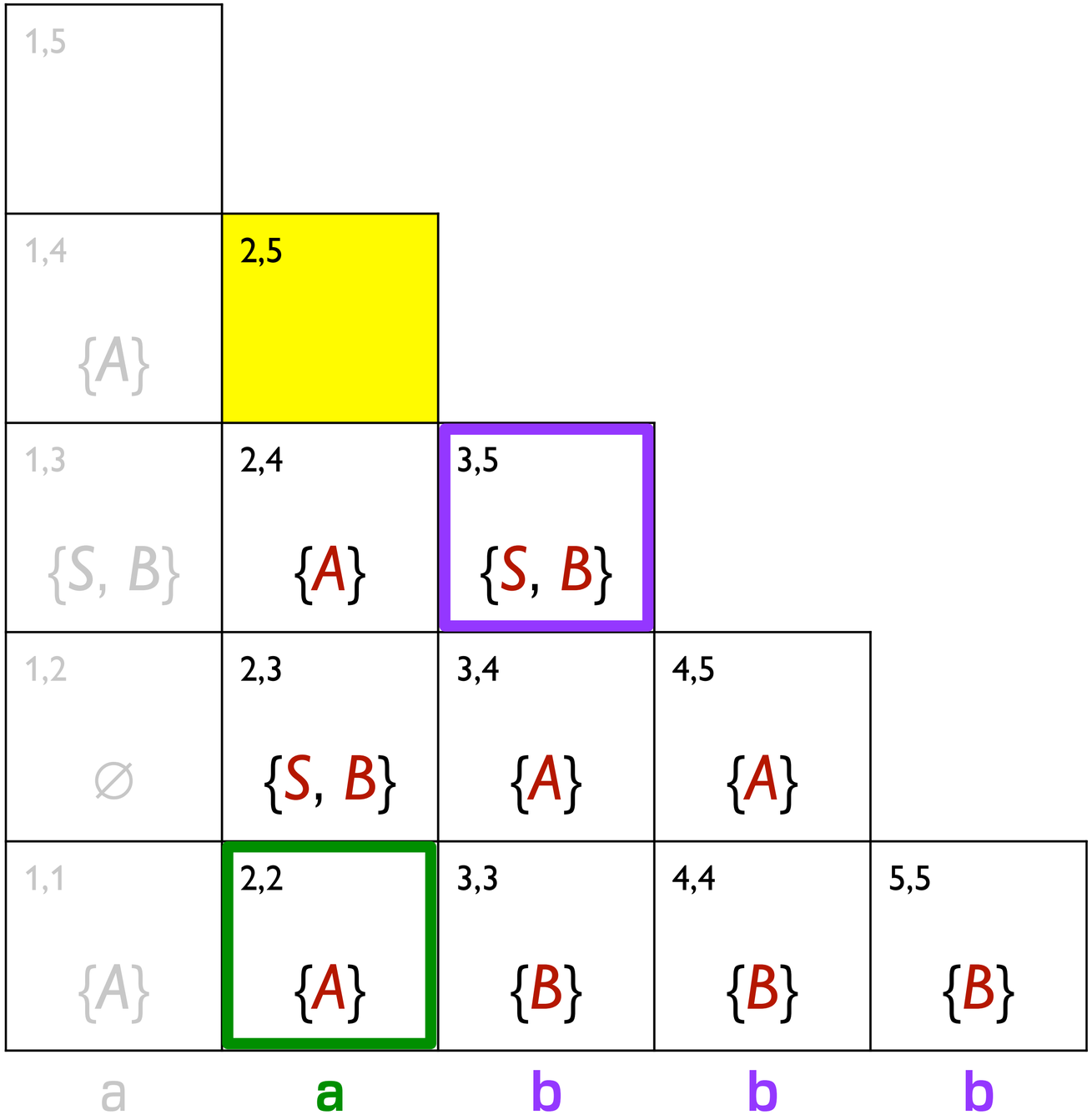
$B \rightarrow AB \mid b$

1,5				
1,4 $\{A\}$	2,5			
1,3 $\{S, B\}$	2,4 $\{A\}$	3,5 $\{S, B\}$		
1,2 $\varnothing$	2,3 $\{S, B\}$	3,4 $\{A\}$	4,5 $\{A\}$	
1,1 $\{A\}$	2,2 $\{A\}$	3,3 $\{B\}$	4,4 $\{B\}$	5,5 $\{B\}$
a	**a**	**b**	**b**	**b**

$S \rightarrow AB$

$A \rightarrow BB \mid a$

$B \rightarrow AB \mid b$



$S \rightarrow AB$

$A \rightarrow BB \mid a$

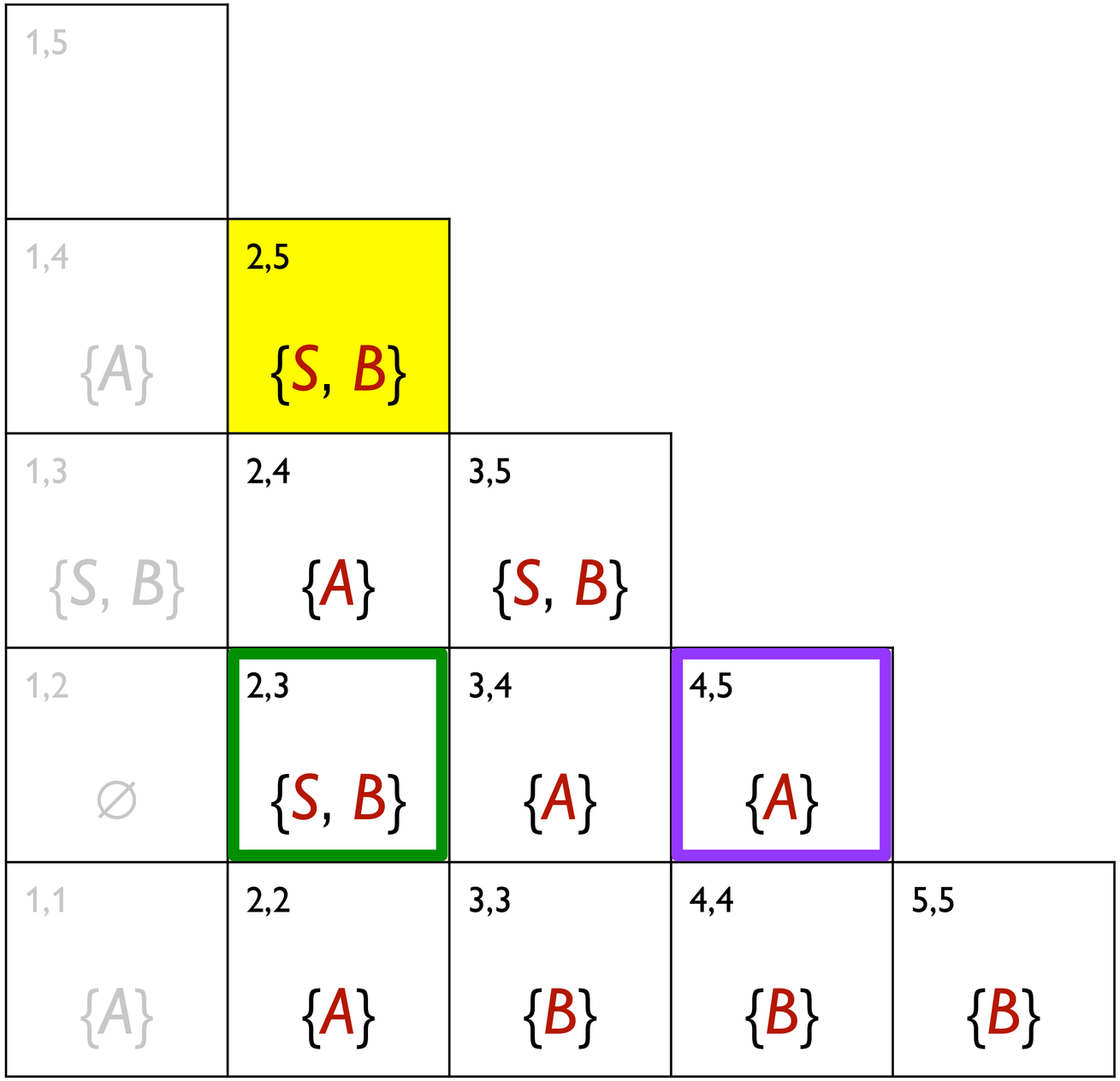
$B \rightarrow AB \mid b$

1,5				
1,4 $\{A\}$	2,5 $\{S, B\}$			
1,3 $\{S, B\}$	2,4 $\{A\}$	3,5 $\{S, B\}$		
1,2 $\varnothing$	2,3 $\{S, B\}$	3,4 $\{A\}$	4,5 $\{A\}$	
1,1 $\{A\}$	2,2 $\{A\}$	3,3 $\{B\}$	4,4 $\{B\}$	5,5 $\{B\}$
a	**a**	**b**	**b**	**b**

$S \rightarrow AB$

$A \rightarrow BB \mid a$

$B \rightarrow AB \mid b$



$S \rightarrow AB$

$A \rightarrow BB \mid a$

$B \rightarrow AB \mid b$

1,5				
1,4 $\{A\}$	2,5 $\{S, B\}$			
1,3 $\{S, B\}$	2,4 $\{A\}$	3,5 $\{S, B\}$		
1,2 $\varnothing$	2,3 $\{S, B\}$	3,4 $\{A\}$	4,5 $\{A\}$	
1,1 $\{A\}$	2,2 $\{A\}$	3,3 $\{B\}$	4,4 $\{B\}$	5,5 $\{B\}$
a	**a**	**b**	**b**	**b**

$S \rightarrow AB$

$A \rightarrow BB \mid a$

$B \rightarrow AB \mid b$

1,5				
1,4 $\{A\}$	2,5 $\{S, B\}$			
1,3 $\{S, B\}$	2,4 $\{A\}$	3,5 $\{S, B\}$		
1,2 $\varnothing$	2,3 $\{S, B\}$	3,4 $\{A\}$	4,5 $\{A\}$	
1,1 $\{A\}$	2,2 $\{A\}$	3,3 $\{B\}$	4,4 $\{B\}$	5,5 $\{B\}$
a	**a**	**b**	**b**	**b**

$S \rightarrow AB$

$A \rightarrow BB \mid a$

$B \rightarrow AB \mid b$

1,5				
1,4 $\{A\}$	2,5 $\{S, B\}$			
1,3 $\{S, B\}$	2,4 $\{A\}$	3,5 $\{S, B\}$		
1,2 $\varnothing$	2,3 $\{S, B\}$	3,4 $\{A\}$	4,5 $\{A\}$	
1,1 $\{A\}$	2,2 $\{A\}$	3,3 $\{B\}$	4,4 $\{B\}$	5,5 $\{B\}$
a	a	b	b	b

$S \rightarrow AB$

$A \rightarrow BB \mid a$

$B \rightarrow AB \mid b$

1,5				
1,4 $\{A\}$	2,5 $\{S, B\}$			
1,3 $\{S, B\}$	2,4 $\{A\}$	3,5 $\{S, B\}$		
1,2 $\varnothing$	2,3 $\{S, B\}$	3,4 $\{A\}$	4,5 $\{A\}$	
1,1 $\{A\}$	2,2 $\{A\}$	3,3 $\{B\}$	4,4 $\{B\}$	5,5 $\{B\}$
a	a	b	b	b

$S \rightarrow AB$

$A \rightarrow BB \mid a$

$B \rightarrow AB \mid b$

1,5				
1,4 $\{A\}$	2,5 $\{S, B\}$			
1,3 $\{S, B\}$	2,4 $\{A\}$	3,5 $\{S, B\}$		
1,2 $\varnothing$	2,3 $\{S, B\}$	3,4 $\{A\}$	4,5 $\{A\}$	
1,1 $\{A\}$	2,2 $\{A\}$	3,3 $\{B\}$	4,4 $\{B\}$	5,5 $\{B\}$
a	**a**	**b**	**b**	**b**

$S \rightarrow AB$

$A \rightarrow BB \mid a$

$B \rightarrow AB \mid b$

1,5 $\{S, B\}$				
1,4 $\{A\}$	2,5 $\{S, B\}$			
1,3 $\{S, B\}$	2,4 $\{A\}$	3,5 $\{S, B\}$		
1,2 $\varnothing$	2,3 $\{S, B\}$	3,4 $\{A\}$	4,5 $\{A\}$	
1,1 $\{A\}$	2,2 $\{A\}$	3,3 $\{B\}$	4,4 $\{B\}$	5,5 $\{B\}$
a	**a**	**b**	**b**	**b**

$S \rightarrow AB$

$A \rightarrow BB \mid a$

$B \rightarrow AB \mid b$

1,5 $\{S, B\}$				
1,4 $\{A\}$	2,5 $\{S, B\}$			
1,3 $\{S, B\}$	2,4 $\{A\}$	3,5 $\{S, B\}$		
1,2 $\varnothing$	2,3 $\{S, B\}$	3,4 $\{A\}$	4,5 $\{A\}$	
1,1 $\{A\}$	2,2 $\{A\}$	3,3 $\{B\}$	4,4 $\{B\}$	5,5 $\{B\}$
a	**a**	**b**	**b**	**b**

$S \rightarrow AB$

$A \rightarrow BB \mid a$

$B \rightarrow AB \mid b$

1,5 $\{S, B\}$				
1,4 $\{A\}$	2,5 $\{S, B\}$			
1,3 $\{S, B\}$	2,4 $\{A\}$	3,5 $\{S, B\}$		
1,2 $\varnothing$	2,3 $\{S, B\}$	3,4 $\{A\}$	4,5 $\{A\}$	
1,1 $\{A\}$	2,2 $\{A\}$	3,3 $\{B\}$	4,4 $\{B\}$	5,5 $\{B\}$
a	**a**	**b**	**b**	**b**

$S \rightarrow AB$

$A \rightarrow BB \mid a$

$B \rightarrow AB \mid b$

1,5 $\{S, B\}$				
1,4 $\{A\}$	2,5 $\{S, B\}$			
1,3 $\{S, B\}$	2,4 $\{A\}$	3,5 $\{S, B\}$		
1,2 $\varnothing$	2,3 $\{S, B\}$	3,4 $\{A\}$	4,5 $\{A\}$	
1,1 $\{A\}$	2,2 $\{A\}$	3,3 $\{B\}$	4,4 $\{B\}$	5,5 $\{B\}$
a	**a**	**b**	**b**	**b**

$S \rightarrow AB$

$A \rightarrow BB \mid a$

$B \rightarrow AB \mid b$

1,5 $\{S, B\}$				
1,4 $\{A\}$	2,5 $\{S, B\}$			
1,3 $\{S, B\}$	2,4 $\{A\}$	3,5 $\{S, B\}$		
1,2 $\varnothing$	2,3 $\{S, B\}$	3,4 $\{A\}$	4,5 $\{A\}$	
1,1 $\{A\}$	2,2 $\{A\}$	3,3 $\{B\}$	4,4 $\{B\}$	5,5 $\{B\}$
a	**a**	**b**	**b**	**b**

$S \rightarrow AB$

$A \rightarrow BB \mid a$

$B \rightarrow AB \mid b$

1,5 $\{S, B\}$				
1,4 $\{A\}$	2,5 $\{S, B\}$			
1,3 $\{S, B\}$	2,4 $\{A\}$	3,5 $\{S, B\}$		
1,2 $\varnothing$	2,3 $\{S, B\}$	3,4 $\{A\}$	4,5 $\{A\}$	
1,1 $\{A\}$	2,2 $\{A\}$	3,3 $\{B\}$	4,4 $\{B\}$	5,5 $\{B\}$
a	a	b	b	b

$S \rightarrow AB$

$A \rightarrow BB \mid a$

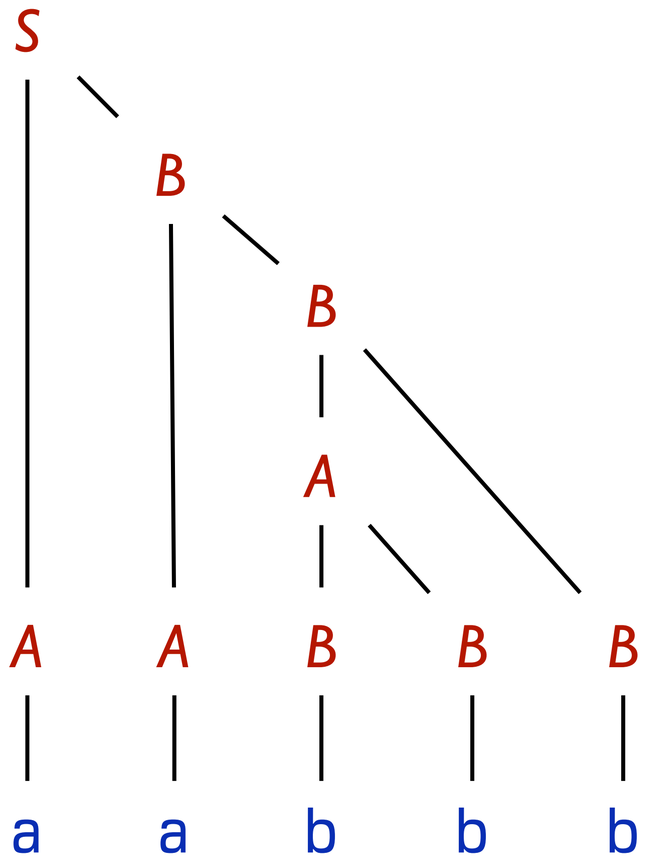
$B \rightarrow AB \mid b$

Since $S \in X_{1,5}$, $w \in L(G)$

1,5 $\{S, B\}$				
1,4 $\{A\}$	2,5 $\{S, B\}$			
1,3 $\{S, B\}$	2,4 $\{A\}$	3,5 $\{S, B\}$		
1,2 $\varnothing$	2,3 $\{S, B\}$	3,4 $\{A\}$	4,5 $\{A\}$	
1,1 $\{A\}$	2,2 $\{A\}$	3,3 $\{B\}$	4,4 $\{B\}$	5,5 $\{B\}$
a	a	b	b	b

The parse tree for the input is the selection of rules that let us cover the entire string with the start symbol.

1,5 {S, B}				
1,4 {A}	2,5 {S, B}			
1,3 {S, B}	2,4 {A}	3,5 {S, B}		
1,2 ∅	2,3 {S, B}	3,4 {A}	4,5 {A}	
1,1 {A}	2,2 {A}	3,3 {B}	4,4 {B}	5,5 {B}
a	a	b	b	b



CYK as a parsing algorithm

Applicability of the CYK algorithm as a parser is limited by the computational requirements needed to find a derivation.

- For an input string of length n, $(n^2+n)/2$ sets need to be constructed to complete the dynamic programming table.
- Each of these sets may require the consideration of several decompositions of the associated substring.

There are other parsing algorithms we can use!

Putting it all together

Parsing techniques like CYK get used with grammars for programming languages – and human languages.

Let Σ = {a, Alice, big, cat, chased, dog, saw, the}.

S → *NP VP*
NP → *Det Adj N* | *Name*
VP → *V NP* | *V*
Adj → big | ε
Det → the | a
N → cat | dog
V → chased | saw
Name → Alice

Is Alice saw the big cat generated by this grammar?

Let Σ = {a, Alice, big, cat, chased, dog, saw, the}.

$S \rightarrow NP\ VP$

$NP \rightarrow Det\ Adj\ N \mid Name$

$VP \rightarrow V\ NP \mid V$

$Adj \rightarrow \texttt{big} \mid \varepsilon$

$Det \rightarrow \texttt{the} \mid \texttt{a}$

$N \rightarrow \texttt{cat} \mid \texttt{dog}$

$V \rightarrow \texttt{chased} \mid \texttt{saw}$

$Name \rightarrow \texttt{Alice}$

Step 1: Convert to CNF

1. *Make start symbol non-recursive*
2. *Eliminate all ε-rules (except $S \rightarrow \varepsilon$)*
3. *Eliminate all chain/unit rules*
4. *Eliminate useless symbols*
5. *Split rules if necessary*

Is `Alice saw the big cat` generated by this grammar?

Let Σ = {a, Alice, big, cat, chased, dog, saw, the}.

S → *NP VP*
NP → *Det X* | *Det N* | Alice
X → *Adj N*
VP → *V NP* | chased | saw
Adj → big
Det → the | a
N → cat | dog
V → chased | saw

Is Alice saw the big cat generated by this grammar?

1,5				
1,4	2,5			
1,3	2,4	3,5		
1,2	2,3	3,4	4,5	
1,1	2,2	3,3	4,4	5,5
Alice	saw	the	big	cat

Step 2: CYK parsing

1,5 {S}				
1,4 ∅	2,5 {VP}			
1,3 ∅	2,4 ∅	3,5 {NP}		
1,2 {S}	2,3 ∅	3,4 ∅	4,5 {X}	
1,1 {NP}	2,2 {V, VP}	3,3 {Det}	4,4 {Adj}	5,5 {N}
Alice	saw	the	big	cat

Extra practice: `if` statements

Let Σ = {`if`, `elif`, `else:`, `:`, `True`, `False`}.

Expr → `if` *Bool* `:` *Expr Mid End* | *Bool*
Mid → `elif` *Bool* `:` *Expr Mid* | ε
End → `else:` *Expr* | ε
Bool → `True` | `False`

Is `if True : False else: True` generated by this grammar?

Let Σ = {`if`, `elif`, `else:`, `:`, `True`, `False`}.

Expr → `if` *Bool* `:` *Expr Mid End* | *Bool*

Mid → `elif` *Bool* `:` *Expr Mid* | ε

End → `else:` *Expr* | ε

Bool → `True` | `False`

To make this more manageable, let's remove the rules about **`elif`**.

Is `if True : False else: True` generated by this grammar?

Let Σ = {`if`, `else:`, `:`, `True`, `False`}.

Expr → `if` *Bool* `:` *Expr End* | *Bool*

End → `else:` *Expr* | ε

Bool → `True` | `False`

Step 1: Convert to CNF

1. *Make start symbol non-recursive*
2. *Eliminate all ε-rules (except S → ε)*
3. *Eliminate all chain/unit rules*
4. *Eliminate useless symbols*
5. *Split rules if necessary*

Is `if True : False else: True` generated by this grammar?

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

Step 2: CYK parsing

```
if      True      :      False     else:     True
```

Appendix: CNF conversion details

Conversion to CNF

1. Make start symbol non-recursive
2. Eliminate all ε-rules (except $S \to \varepsilon$)
3. Eliminate all chain/unit rules (e.g., $A \to B$)
4. Eliminate useless symbols
5. Split rules if necessary

Non-recursive start symbol

The first goal is to limit the role of the start symbol S to the initiation of a derivation.

To produce a CFG G' that is equivalent to G but does not have S in the body of any rules:

If S does not appear in the body of any rule, then $G' = G$

Otherwise, add a new start variable S' and the rule $S' \to S$

Eliminating ε-rules

We want to eliminate variables that don't generate terminal symbols.

Advantage: Reduces the length of derivations

Disadvantage: Increases the number of rules

This requires the identification of a set of *nullable* variables, i.e., variables that can derive the empty string by a sequence of rule applications, $A \stackrel{*}{\Rightarrow} \varepsilon$.

Eliminating ε-rules: Finding nullable variables

1. Put each variable A for which a rule $A \to \varepsilon$ exists in the set *NULLABLE*.

2. Repeat:

   For each variable $A \in V$:

   If $A \to A_1 A_2 \ldots A_k$ is a rule where each A_i is in *NULLABLE*, then add A to *NULLABLE*

   If nothing was added to *NULLABLE* in this iteration, stop.

Eliminating ε-rules

1. Construct the set of nullable variables, as described.
2. If the start symbol S is in the nullable set, add the rule $S \rightarrow \varepsilon$.
3. Consider each rule $A \rightarrow \omega$, where ω contains a sequence of n nullable variables, i.e., $\omega = \omega_1 A_1 \omega_2 A_2 \ldots \omega_n A_n \omega_{n+1}$, where each ω_i is a sequence of terminals and non-nullable variables.

   Replace $A \rightarrow \omega$ with a set of 2^n rules, where each new rule contains the original sequence of terminals and non-nullable variables combined with all possible combinations of the nullable variables.
4. Remove all $A \rightarrow \varepsilon$ rules, except $S \rightarrow \varepsilon$.

Eliminating ε-rules: Example

Consider the rule $A \to BABa$ and suppose B is a nullable variable.

Since there are two nullable variables in the rule body, we must replace this one rule with the original rule and three new rules (total: $2^2 = 4$ rules):

$A \to BABa$

$A \to ABa$

$A \to BAa$

$A \to Aa$

Eliminating chain rules

A rule of the form $A \to B$ is called a chain rule or a unit production.

These rules are just renaming variables in sentential forms; they mean the same strings that are derivable from B are also derivable from A.

Since chain rules do not make progress toward deriving a terminal string, we can remove them from the CFG.

More rules are added in the process, to make sure the same language is derivable.

Eliminating chain rules

Another step to reduce the length of derivations (while possibly increasing the number of rules):

For each variable A:

Compute $CHAIN(A)$, the set of all variables derivable from A by chain rules.

For each non-chain rule $B \rightarrow \omega$:

If $B \in CHAIN(A)$, replace rule $A \rightarrow B$ with $A \rightarrow \omega$

Computing *CHAIN*(*A*)

CHAIN(*A*) = {*A*}

Repeat:

For each variable *B* added to *CHAIN*(*A*) last cycle:

For each rule *B* → *C*:

Add *C* to *CHAIN*(*A*)

If *CHAIN*(*A*) was not changed in this iteration, stop.

Eliminating useless symbols

A variable is *useless* if it never appears in the derivation of a string. To eliminate useless symbols:

1. *Eliminate variables that cannot derive terminal strings.* To do this, find all variables that *can* derive a terminal string.

   TERM = {A | $A \rightarrow$ *string of terminals* is a rule}

   Repeat:

   For each $A \in$ *TERM*:

   If $A \rightarrow \omega$ is a rule, where ω is a string of terminals and variables in *TERM*:

   Add A to *TERM*

   If no more variables were added to *TERM* in this iteration, stop.

   Then eliminate:

   a. variables not in *TERM*

   b. rules involving a variable not in *TERM*

Eliminating useless symbols

2. *Eliminate symbols that are not reachable from S.* To do this, find all variables that are reachable:

    REACH = {*S*}

    Repeat:

        For each *A* just added to *REACH*:

            For each rule *A* → ω:

                Add all variables in ω to *REACH*

        If *REACH* was not changed, stop.

    Then eliminate:

    a. variables not in *REACH*

    b. rules involving a variable not in *REACH*

    c. all terminals not in remaining rules

At this point, all rules are of the form

$S \rightarrow \varepsilon$

$A \rightarrow a$

$A \rightarrow \omega$, where ω is a string of more than one terminals, variables, or terminals and variables (excluding S)

At this point, all rules are of the form

$S \rightarrow \varepsilon$

$A \rightarrow a$

$A \rightarrow \omega$, where ω is a string of more than one terminals, variables, or terminals and variables (excluding S)

But Chomsky normal form is even stricter, allowing only:

$S \rightarrow \varepsilon$

$A \rightarrow a$

$A \rightarrow BC$, where neither $B \neq S$ and $C \neq S$

Split rules

Step 1: For each rule $A \to \omega$, replace each terminal a in ω with a new variable V_a. Add the rule $V_a \to a$.

E.g.,

$A \to bDcF$

becomes

$A \to V_b D V_c F$

$V_b \to b$

$V_c \to c$

Split rules

Step 2: Now introduce new variables to remove variables one by one to get only two variables in the body of each rule.

E.g.,

$A \rightarrow BCDF$

becomes

$A \rightarrow BT_1$

$T_1 \rightarrow CT_2$

$T_2 \rightarrow DF$

Acknowledgments

This lecture incorporates material from:

John Hopcroft and Jeffrey Ullman, *Introduction to Automata Theory, Languages and Computation*

Nancy Ide, Vassar College

Michael Sipser, *Introduction to the Theory of Computation*