Weighted Parsing, Probabilistic Parsing

Our bane: Ambiguity

- John saw Mary
  - Typhoid Mary
  - Phillips screwdriver Mary

  note how rare rules interact
- I see a bird
  - is this 4 nouns – parsed like “city park scavenger bird”?

  rare parts of speech, plus systematic ambiguity in noun sequences
- Time flies like an arrow
  - Fruit flies like a banana
  - Time reactions like this one
  - Time reactions like a chemist
  - or is it just an NP?

Our bane: Ambiguity

- John saw Mary
  - Typhoid Mary
  - Phillips screwdriver Mary

  note how rare rules interact
- I see a bird
  - is this 4 nouns – parsed like “city park scavenger bird”?

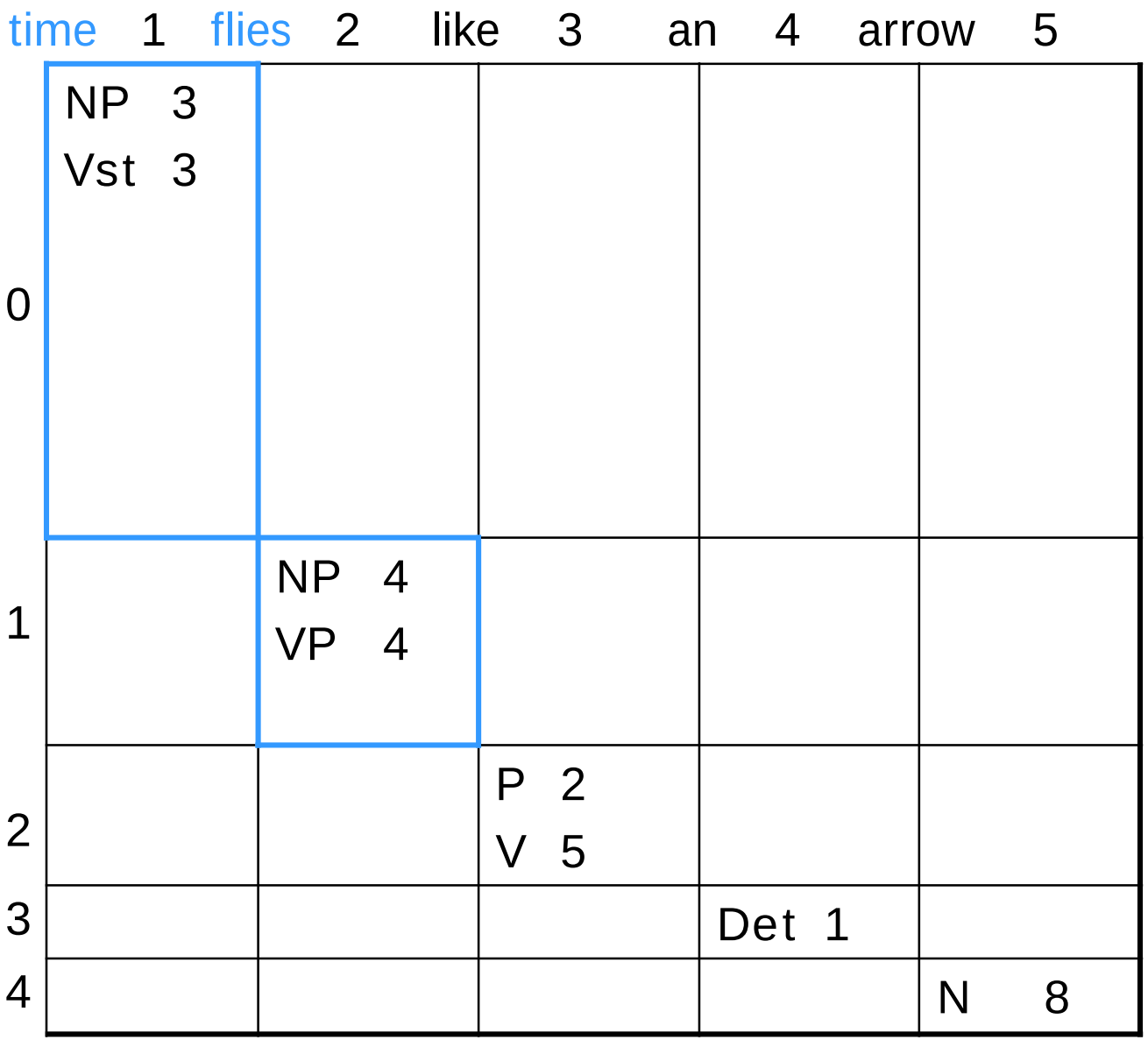
  rare parts of speech, plus systematic ambiguity in noun sequences
- Time | flies like an arrow — NP VP
  - Fruit flies | like a banana — NP VP
  - Time | reactions like this one — V[stem] NP
  - Time reactions | like a chemist — S PP
  - or is it just an NP?

How to solve this combinatorial explosion of ambiguity?

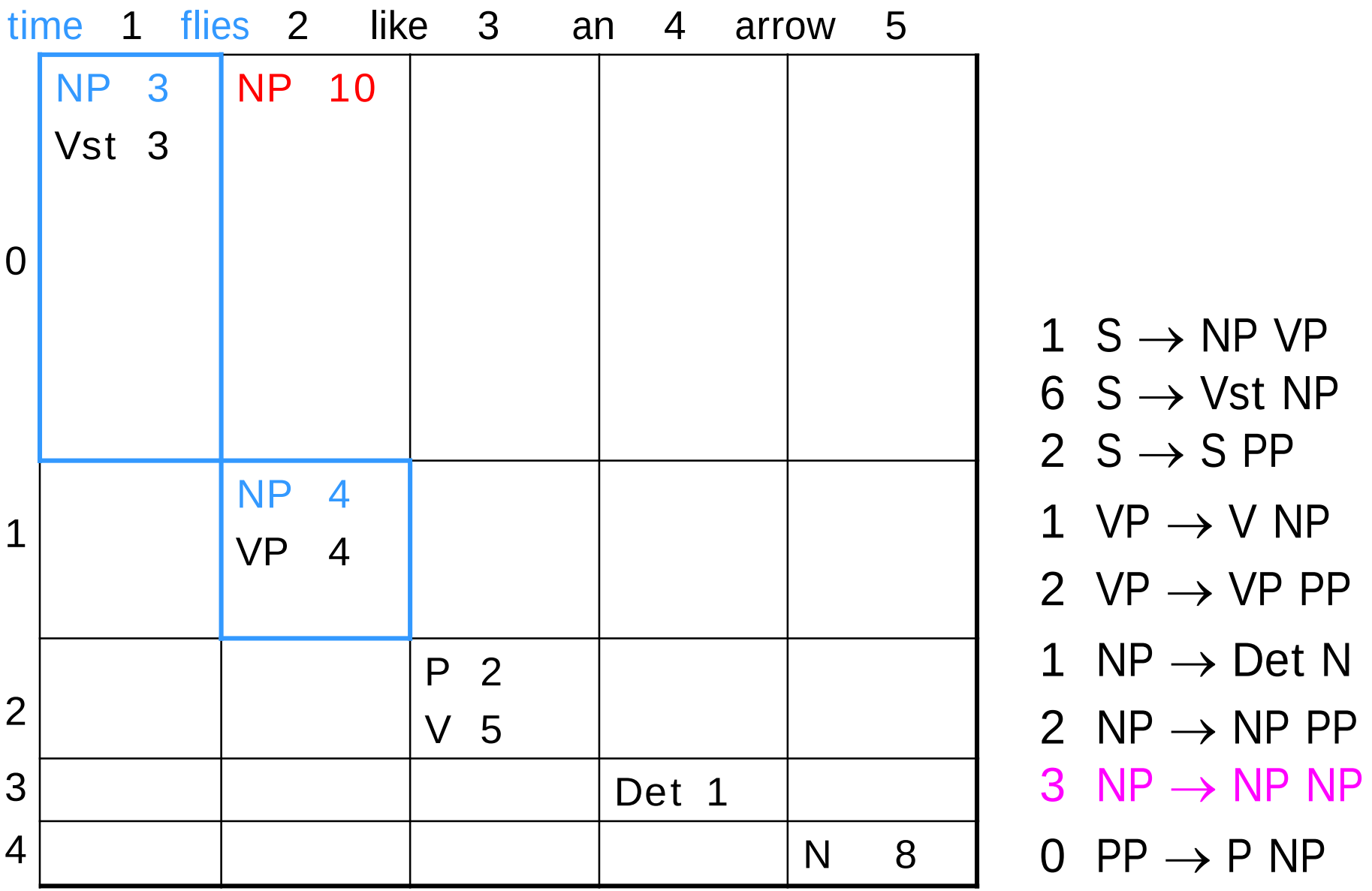
1. First try parsing without any weird rules, throwing them in only if needed.
2. Better: every rule has a weight.
   A tree's weight is total weight of all its rules.
   Pick the overall lightest parse of sentence.
3. Can we pick the weights automatically?
   We'll get to this later ...

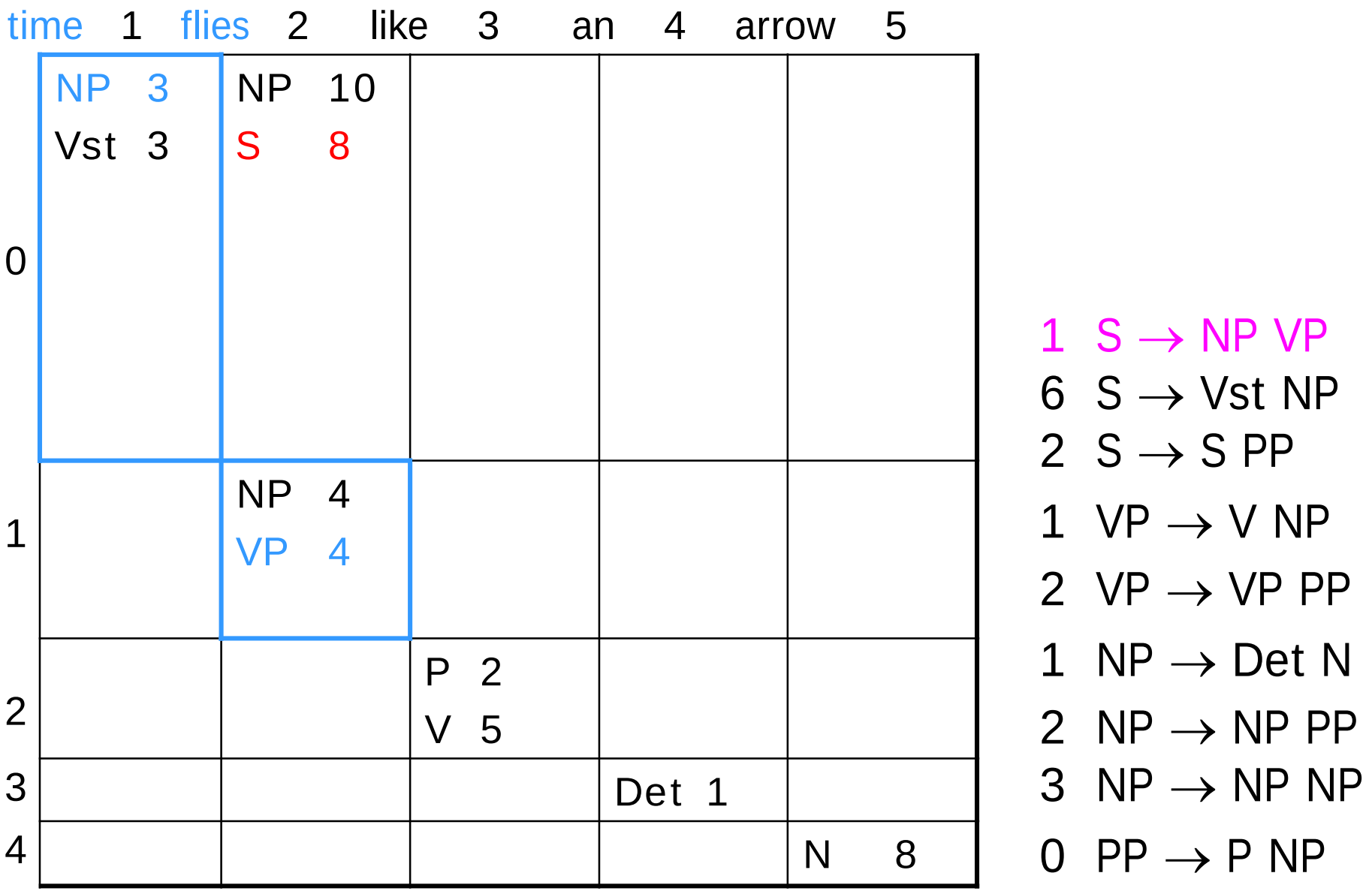
	time 1	flies 2	like 3	an 4	arrow 5
0	NP 3 Vst 3				
1		NP 4 VP 4			
2			P 2 V 5		
3				Det 1	
4					N 8

1 S → NP VP
6 S → Vst NP
2 S → S PP
1 VP → V NP
2 VP → VP PP
1 NP → Det N
2 NP → NP PP
3 NP → NP NP
0 PP → P NP



1 S → NP VP
6 S → Vst NP
2 S → S PP
1 VP → V NP
2 VP → VP PP
1 NP → Det N
2 NP → NP PP
3 NP → NP NP
0 PP → P NP


time 1
flies 2
like 3
an 4
arrow 5
0
1
2
3
4
NP 3
Vst 3
NP 10
NP 4
VP 4
P 2
V 5
Det 1
N 8
1 S → NP VP
6 S → Vst NP
2 S → S PP
1 VP → V NP
2 VP → VP PP
1 NP → Det N
2 NP → NP PP
3 NP → NP NP
0 PP → P NP


time 1
flies 2
like 3
an 4
arrow 5
0
1
2
3
4
NP 3
Vst 3
NP 10
S 8
NP 4
VP 4
P 2
V 5
Det 1
N 8
1 S → NP VP
6 S → Vst NP
2 S → S PP
1 VP → V NP
2 VP → VP PP
1 NP → Det N
2 NP → NP PP
3 NP → NP NP
0 PP → P NP

	time 1	flies 2	like 3	an 4	arrow 5
0	NP 3 Vst 3	NP 10 S 8 S 13			
1		NP 4 VP 4			
2			P 2 V 5		
3				Det 1	
4					N 8

1 S → NP VP
6 S → Vst NP
2 S → S PP
1 VP → V NP
2 VP → VP PP
1 NP → Det N
2 NP → NP PP
3 NP → NP NP
0 PP → P NP

	time 1	flies 2	like 3	an 4	arrow 5
0	NP 3 Vst 3	NP 10 S 8 S 13			
1		NP 4 VP 4			
2			P 2 V 5		
3				Det 1	
4					N 8

1 S → NP VP
6 S → Vst NP
2 S → S PP
1 VP → V NP
2 VP → VP PP
1 NP → Det N
2 NP → NP PP
3 NP → NP NP
0 PP → P NP

	time 1	flies 2	like 3	an 4	arrow 5
0	NP 3 Vst 3	NP 10 S 8 S 13			
1		NP 4 VP 4			
2			P 2 V 5		
3				Det 1	NP 10
4					N 8

1 S → NP VP
6 S → Vst NP
2 S → S PP
1 VP → V NP
2 VP → VP PP
1 NP → Det N
2 NP → NP PP
3 NP → NP NP
0 PP → P NP

	time 1	flies 2	like 3	an 4	arrow 5
0	NP 3 Vst 3	NP 10 S 8 S 13			
1		NP 4 VP 4			
2			P 2 V 5		
3				Det 1	NP 10
4					N 8

1 S → NP VP
6 S → Vst NP
2 S → S PP
1 VP → V NP
2 VP → VP PP
1 NP → Det N
2 NP → NP PP
3 NP → NP NP
0 PP → P NP

	time 1	flies 2	like 3	an 4	arrow 5
0	NP 3 Vst 3	NP 10 S 8 S 13			
1		NP 4 VP 4			
2			P 2 V 5		PP 12
3				Det 1	NP 10
4					N 8

1 S → NP VP
6 S → Vst NP
2 S → S PP
1 VP → V NP
2 VP → VP PP
1 NP → Det N
2 NP → NP PP
3 NP → NP NP
0 PP → P NP

	time 1	flies 2	like 3	an 4	arrow 5
0	NP 3 Vst 3	NP 10 S 8 S 13			
1		NP 4 VP 4			
2			P 2 V 5		PP 12 VP 16
3				Det 1	NP 10
4					N 8

1 S → NP VP
6 S → Vst NP
2 S → S PP
1 VP → V NP
2 VP → VP PP
1 NP → Det N
2 NP → NP PP
3 NP → NP NP
0 PP → P NP

time 1	flies 2	like 3	an 4	arrow 5
0	NP 3 Vst 3	NP 10 S 8 S 13			
1		NP 4 VP 4			
2			P 2 V 5		PP 12 VP 16
3				Det 1	NP 10
4					N 8

1 S → NP VP
6 S → Vst NP
2 S → S PP
1 VP → V NP
2 VP → VP PP
1 NP → Det N
2 NP → NP PP
3 NP → NP NP
0 PP → P NP

time	1 flies	2 like	3 an	4 arrow	5
0	NP 3 Vst 3	NP 10 S 8 S 13			
1		NP 4 VP 4			NP 18
2			P 2 V 5		PP 12 VP 16
3				Det 1	NP 10
4					N 8

1 S → NP VP
6 S → Vst NP
2 S → S PP
1 VP → V NP
2 VP → VP PP
1 NP → Det N
2 NP → NP PP
3 NP → NP NP
0 PP → P NP

time	1	flies	2	like	3	an	4	arrow	5
0	NP 3 Vst 3		NP 10 S 8 S 13						
1			NP 4 VP 4					NP 18 S 21	
2					P 2 V 5			PP 12 VP 16	
3							Det 1	NP 10	
4								N 8	

1 S → NP VP
6 S → Vst NP
2 S → S PP
1 VP → V NP
2 VP → VP PP
1 NP → Det N
2 NP → NP PP
3 NP → NP NP
0 PP → P NP

time 1	flies 2	like 3	an 4	arrow 5
0	NP 3 Vst 3	NP 10 S 8 S 13			
1		NP 4 VP 4			NP 18 S 21 VP 18
2			P 2 V 5		PP 12 VP 16
3				Det 1	NP 10
4					N 8

1 S → NP VP
6 S → Vst NP
2 S → S PP
1 VP → V NP
2 VP → VP PP
1 NP → Det N
2 NP → NP PP
3 NP → NP NP
0 PP → P NP

	time 1	flies 2	like 3	an 4	arrow 5
0	NP 3 Vst 3	NP 10 S 8 S 13			
1		NP 4 VP 4			NP 18 S 21 VP 18
2			P 2 V 5		PP 12 VP 16
3				Det 1	NP 10
4					N 8

1 S → NP VP
6 S → Vst NP
2 S → S PP
1 VP → V NP
2 VP → VP PP
1 NP → Det N
2 NP → NP PP
3 NP → NP NP
0 PP → P NP

	time 1	flies 2	like 3	an 4	arrow 5
0	NP 3 Vst 3	NP 10 S 8 S 13			NP 24
1		NP 4 VP 4			NP 18 S 21 VP 18
2			P 2 V 5		PP 12 VP 16
3				Det 1	NP 10
4					N 8

1 S → NP VP
6 S → Vst NP
2 S → S PP
1 VP → V NP
2 VP → VP PP
1 NP → Det N
2 NP → NP PP
3 NP → NP NP
0 PP → P NP

	time 1	flies 2	like 3	an 4	arrow 5
0	NP 3 Vst 3	NP 10 S 8 S 13			NP 24 S 22
1		NP 4 VP 4			NP 18 S 21 VP 18
2			P 2 V 5		PP 12 VP 16
3				Det 1	NP 10
4					N 8

1 S → NP VP
6 S → Vst NP
2 S → S PP
1 VP → V NP
2 VP → VP PP
1 NP → Det N
2 NP → NP PP
3 NP → NP NP
0 PP → P NP

	time 1	flies 2	like 3	an 4	arrow 5
0	NP 3 Vst 3	NP 10 S 8 S 13			NP 24 S 22 S 27
1		NP 4 VP 4			NP 18 S 21 VP 18
2			P 2 V 5		PP 12 VP 16
3				Det 1	NP 10
4					N 8

1 S → NP VP

6 S → Vst NP

2 S → S PP

1 VP → V NP

2 VP → VP PP

1 NP → Det N

2 NP → NP PP

3 NP → NP NP

0 PP → P NP

	time 1	flies 2	like 3	an 4	arrow 5
0	NP 3 Vst 3	NP 10 S 8 S 13			NP 24 S 22 S 27
1		NP 4 VP 4			NP 18 S 21 VP 18
2			P 2 V 5		PP 12 VP 16
3				Det 1	NP 10
4					N 8

1 S → NP VP
6 S → Vst NP
2 S → S PP
1 VP → V NP
2 VP → VP PP
1 NP → Det N
2 NP → NP PP
3 NP → NP NP
0 PP → P NP

	time 1	flies 2	like 3	an 4	arrow 5
0	NP 3 Vst 3	NP 10 S 8 S 13			NP 24 S 22 S 27 NP 24
1		NP 4 VP 4			NP 18 S 21 VP 18
2			P 2 V 5		PP 12 VP 16
3				Det 1	NP 10
4					N 8

1 S → NP VP
6 S → Vst NP
2 S → S PP
1 VP → V NP
2 VP → VP PP
1 NP → Det N
2 NP → NP PP
3 NP → NP NP
0 PP → P NP

	time 1	flies 2	like 3	an 4	arrow 5
0	NP 3 Vst 3	NP 10 S 8 S 13			NP 24 S 22 S 27 NP 24 S 27
1		NP 4 VP 4			NP 18 S 21 VP 18
2			P 2 V 5		PP 12 VP 16
3				Det 1	NP 10
4					N 8

1 S → NP VP
6 S → Vst NP
2 S → S PP
1 VP → V NP
2 VP → VP PP
1 NP → Det N
2 NP → NP PP
3 NP → NP NP
0 PP → P NP

	time 1	flies 2	like 3	an 4	arrow 5
0	NP 3 Vst 3	NP 10 S 8 S 13			NP 24 S 22 S 27 NP 24 S 27 S 22
1		NP 4 VP 4			NP 18 S 21 VP 18
2			P 2 V 5		PP 12 VP 16
3				Det 1	NP 10
4					N 8

1 S → NP VP
6 S → Vst NP
2 S → S PP
1 VP → V NP
2 VP → VP PP
1 NP → Det N
2 NP → NP PP
3 NP → NP NP
0 PP → P NP

	time 1	flies 2	like 3	an 4	arrow 5
0	NP 3 Vst 3	NP 10 S 8 S 13			NP 24 S 22 S 27 NP 24 S 27 S 22 S 27
1		NP 4 VP 4			NP 18 S 21 VP 18
2			P 2 V 5		PP 12 VP 16
3				Det 1	NP 10
4					N 8

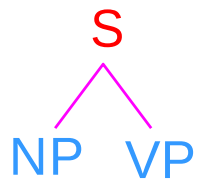
1 S → NP VP
6 S → Vst NP
2 S → S PP
1 VP → V NP
2 VP → VP PP
1 NP → Det N
2 NP → NP PP
3 NP → NP NP
0 PP → P NP

Follow backpointers ...

S

	time 1	flies 2	like 3	an 4	arrow 5
0	NP 3 Vst 3	NP 10 S 8 S 13			NP 24 S 22 S 27 NP 24 S 27 S 22 S 27
1		NP 4 VP 4			NP 18 S 21 VP 18
2			P 2 V 5		PP 12 VP 16
3				Det 1	NP 10
4					N 8

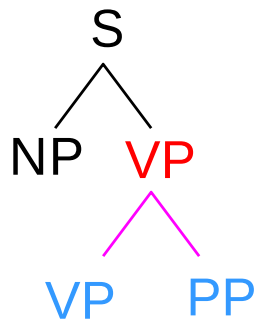
1 S → NP VP
6 S → Vst NP
2 S → S PP
1 VP → V NP
2 VP → VP PP
1 NP → Det N
2 NP → NP PP
3 NP → NP NP
0 PP → P NP



	time 1	flies 2	like 3	an 4	arrow 5
0	NP 3 Vst 3	NP 10 S 8 S 13			NP 24 S 22 S 27 NP 24 S 27 S 22 S 27
1		NP 4 VP 4			NP 18 S 21 VP 18
2			P 2 V 5		PP 12 VP 16
3				Det 1	NP 10
4					N 8

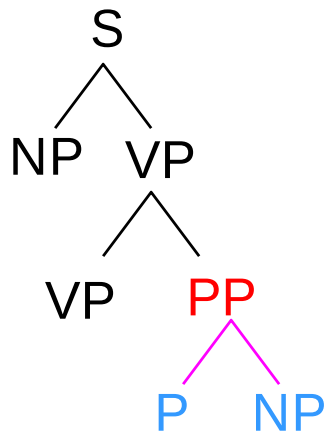
1 S → NP VP
6 S → Vst NP
2 S → S PP
1 VP → V NP
2 VP → VP PP
1 NP → Det N
2 NP → NP PP
3 NP → NP NP
0 PP → P NP

time	1 flies	2 like	3 an	4 arrow	5
0	NP 3 Vst 3	NP 10 S 8 S 13			NP 24 S 22 S 27 NP 24 S 27 S 22 S 27
1		NP 4 VP 4			NP 18 S 21 VP 18
2			P 2 V 5		PP 12 VP 16
3				Det 1	NP 10
4					N 8



1 S → NP VP
6 S → Vst NP
2 S → S PP
1 VP → V NP
2 VP → VP PP
1 NP → Det N
2 NP → NP PP
3 NP → NP NP
0 PP → P NP

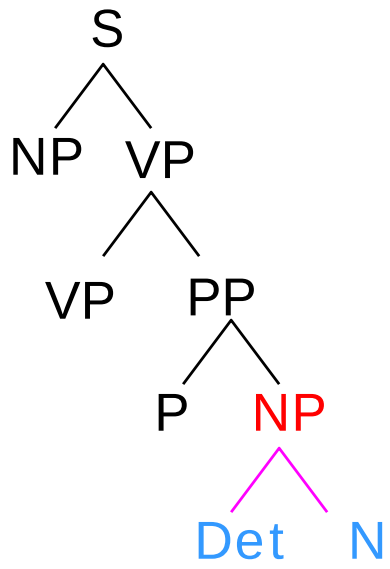
time	1 flies	2 like	3 an	4 arrow	5
0	NP 3 Vst 3	NP 10 S 8 S 13			NP 24 S 22 S 27 NP 24 S 27 S 22 S 27
1		NP 4 VP 4			NP 18 S 21 VP 18
2			P 2 V 5		PP 12 VP 16
3				Det 1	NP 10
4					N 8



1 S → NP VP
6 S → Vst NP
2 S → S PP
1 VP → V NP
2 VP → VP PP
1 NP → Det N
2 NP → NP PP
3 NP → NP NP

0 PP → P NP

time	1 flies	2 like	3 an	4 arrow	5
0	NP 3 Vst 3	NP 10 S 8 S 13			NP 24 S 22 S 27 NP 24 S 27 S 22 S 27
1		NP 4 VP 4			NP 18 S 21 VP 18
2			P 2 V 5		PP 12 VP 16
3				Det 1	NP 10
4					N 8



1 S → NP VP
6 S → Vst NP
2 S → S PP
1 VP → V NP
2 VP → VP PP
1 NP → Det N
2 NP → NP PP
3 NP → NP NP
0 PP → P NP

Which entries do we need?

	time 1	flies 2	like 3	an 4	arrow 5
0	NP 3 Vst 3	NP 10 S 8 S 13			NP 24 S 22 S 27 NP 24 S 27 S 22 S 27
1		NP 4 VP 4			NP 18 S 21 VP 18
2			P 2 V 5		PP 12 VP 16
3				Det 1	NP 10
4					N 8

1 S → NP VP
6 S → Vst NP
2 S → S PP
1 VP → V NP
2 VP → VP PP
1 NP → Det N
2 NP → NP PP
3 NP → NP NP
0 PP → P NP

Which entries do we need?

	time 1	flies 2	like 3	an 4	arrow 5
0	NP 3 Vst 3	NP 10 S 8 S 13			NP 24 S 22 S 27 NP 24 S 27 S 22 S 27
1		NP 4 VP 4			NP 18 S 21 VP 18
2			P 2 V 5		PP 12 VP 16
3				Det 1	NP 10
4					N 8

1 S → NP VP
6 S → Vst NP
2 S → S PP
1 VP → V NP
2 VP → VP PP
1 NP → Det N
2 NP → NP PP
3 NP → NP NP
0 PP → P NP

Not worth keeping ...

	time 1	flies 2	like 3	an 4	arrow 5
0	NP 3 Vst 3	NP 10 S 8 S 13			NP 24 S 22 S 27 NP 24 S 27 S 22 S 27
1		NP 4 VP 4			NP 18 S 21 VP 18
2			P 2 V 5		PP 12 VP 16
3				Det 1	NP 10
4					N 8

1 S → NP VP
6 S → Vst NP
2 S → S PP
1 VP → V NP
2 VP → VP PP
1 NP → Det N
2 NP → NP PP
3 NP → NP NP
0 PP → P NP

... since it just breeds worse options

	time 1	flies 2	like 3	an 4	arrow 5
0	NP 3 Vst 3	NP 10 S 8 S 13			NP 24 S 22 S 27 NP 24 S 27 S 22 S 27
1		NP 4 VP 4			NP 18 S 21 VP 18
2			P 2 V 5		PP 12 VP 16
3				Det 1	NP 10
4					N 8

1 S → NP VP
6 S → Vst NP
2 S → S PP
1 VP → V NP
2 VP → VP PP
1 NP → Det N
2 NP → NP PP
3 NP → NP NP
0 PP → P NP

Keep only best-in-class!

	time 1	flies 2	like 3	an 4	arrow 5
0	NP 3 Vst 3	NP 10 S 8 S 13	inferior stock		NP 24 S 22 S 27 NP 24 S 27 S 22 S 27
1		NP 4 VP 4			NP 18 S 21 VP 18
2			P 2 V 5		PP 12 VP 16
3				Det 1	NP 10
4					N 8

1 S → NP VP
6 S → Vst NP
2 S → S PP
1 VP → V NP
2 VP → VP PP
1 NP → Det N
2 NP → NP PP
3 NP → NP NP
0 PP → P NP

Keep only best-in-class!

(and its backpointers so you can recover best parse)

	time 1	flies 2	like 3	an 4	arrow 5
0	NP 3 Vst 3	NP 10 S 8			NP 24 S 22
1		NP 4 VP 4			NP 18 S 21 VP 18
2			P 2 V 5		PP 12 VP 16
3				Det 1	NP 10
4					N 8

1 S → NP VP
6 S → Vst NP
2 S → S PP
1 VP → V NP
2 VP → VP PP
1 NP → Det N
2 NP → NP PP
3 NP → NP NP
0 PP → P NP

Chart Parsing

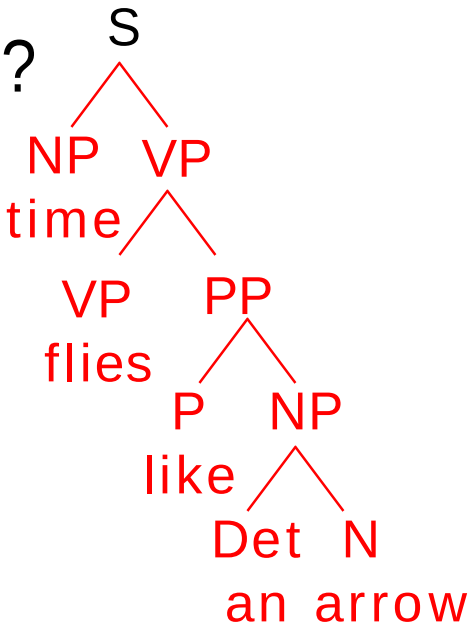
```
phrase(X,I,J)  :-  rewrite(X,W), word(W,I,J).
phrase(X,I,J)  :-  rewrite(X,Y,Z), phrase(Y,I,Mid), phrase(Z,Mid,J).
goal           :-  phrase(start_symbol, 0, sentence_length).
```

Weighted Chart Parsing ("min cost")

```
phrase(X,I,J)  min= rewrite(X,W) + word(W,I,J).
phrase(X,I,J)  min= rewrite(X,Y,Z) + phrase(Y,I,Mid) + phrase(Z,Mid,J).
goal           min=  phrase(start_symbol, 0, sentence_length).
```

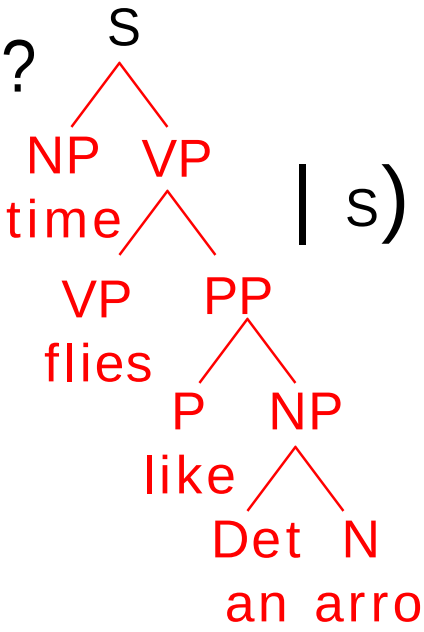
Probabilistic Trees

- Instead of lightest weight tree, take highest probability tree
- Given any tree, your assignment 1 generator would have some probability of producing it!
- Just like using n-grams to choose among strings ...
- What is the probability of this tree?



Probabilistic Trees

- Instead of lightest weight tree, take highest probability tree
- Given any tree, your assignment 1 generator would have some probability of producing it!
- Just like using n-grams to choose among strings ...
- What is the probability of this tree?

p( | S)

- You rolled a lot of independent dice ...

Chain rule: One word at a time

p(time flies like an arrow)
= p(time)
* p(flies | time)
* p(like | time flies)
* p(an | time flies like)
* p(arrow | time flies like an)

Chain rule + backoff (to get trigram model)

p(time flies like an arrow)

= p(time)

* p(flies | time)
* p(like | time flies)
* p(an | time flies like)
* p(arrow | time flies like an)

Chain rule – written differently

p(time flies like an arrow)
= p(time)
* p(time flies | time)
* p(time flies like | time flies)
* p(time flies like an | time flies like)
* p(time flies like an arrow | time flies like an)

Proof: p(x,y | x) = p(x | x) * p(y | x, x) = 1 * p(y | x)

Chain rule + backoff

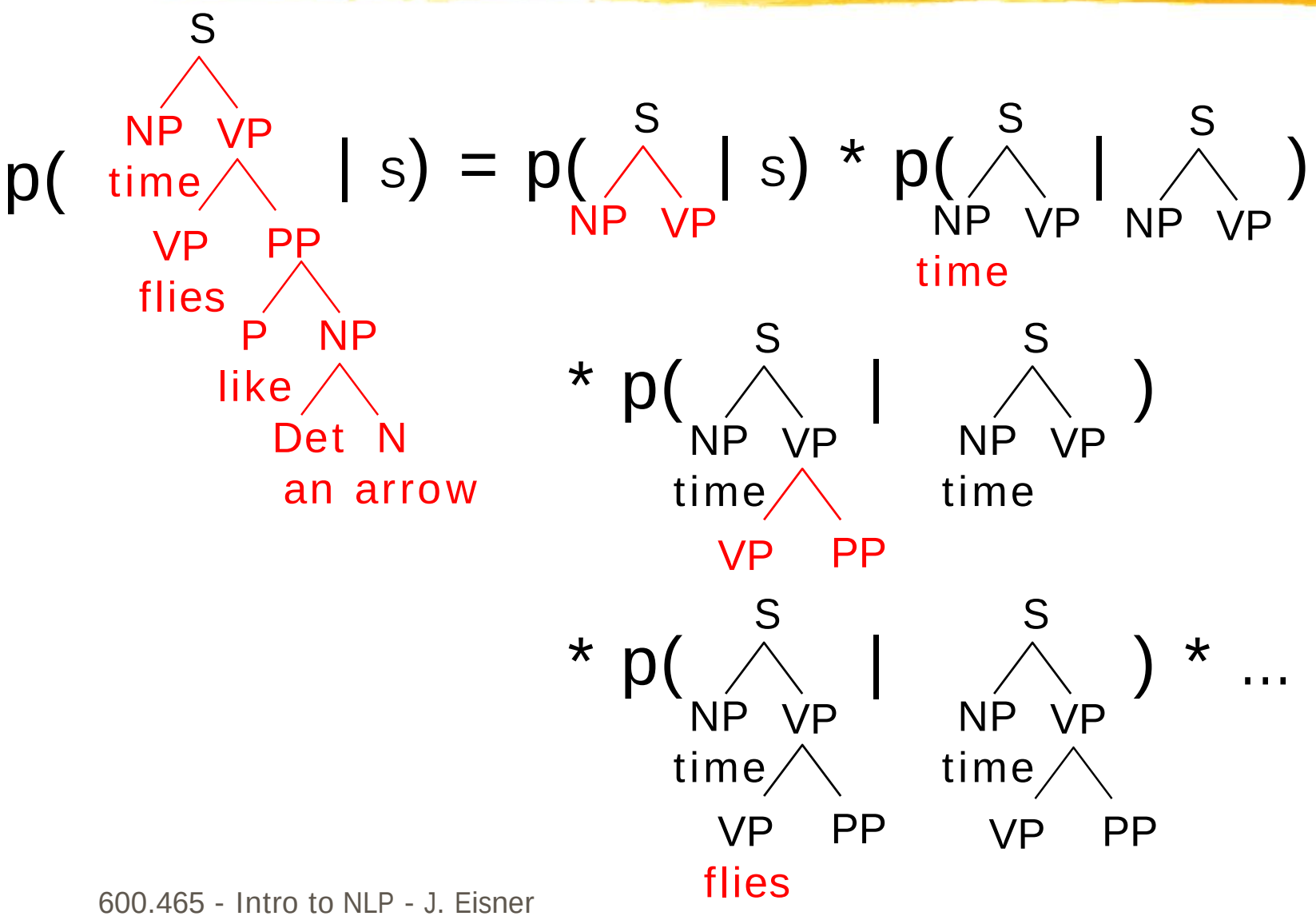
p(time flies like an arrow)

= p(time)

* p(time flies | time)
* p(time flies like | time flies)
* p(time flies like an | time flies like)
* p(time flies like an arrow | time flies like an)

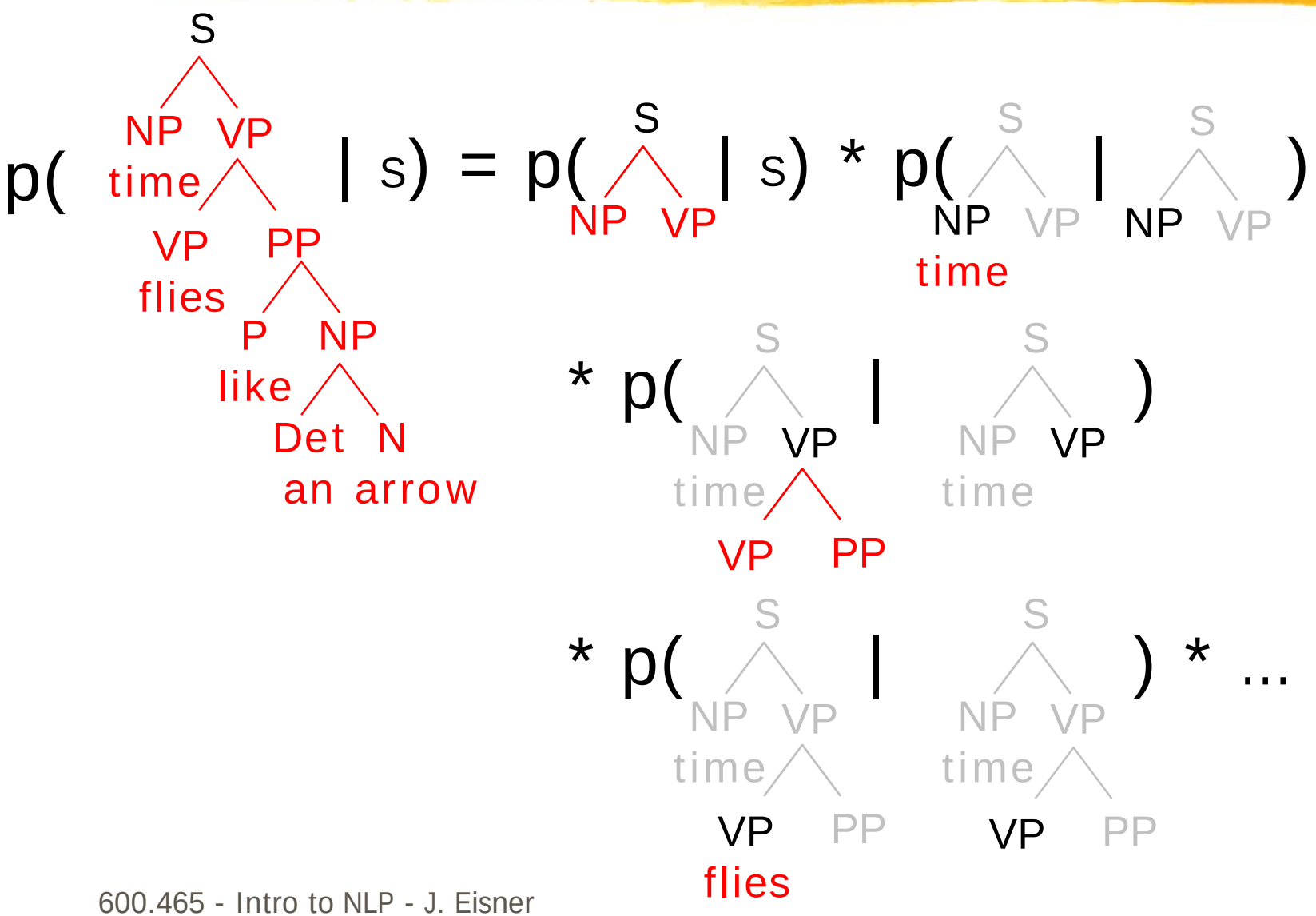
Proof: p(x,y | x) = p(x | x) * p(y | x, x) = 1 * p(y | x)

Chain rule: One node at a time



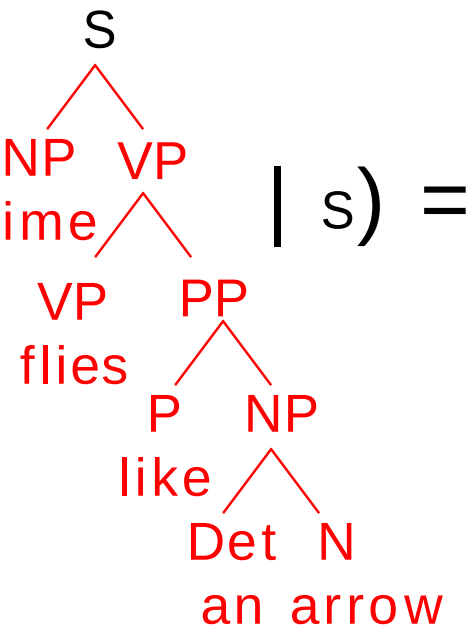
Chain rule + backoff

model you used in homework 1! (called "PCFG")



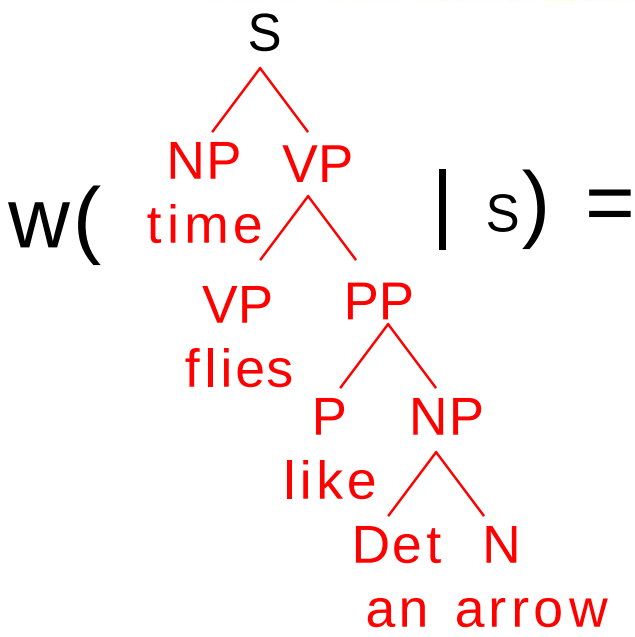
Simplified notation

model you used in homework 1! (called "PCFG")



$$p(\text{tree} \mid S) = p(S \rightarrow NP\ VP \mid S) * p(NP \rightarrow time \mid NP)$$
$$* \ p(VP \rightarrow VP\ NP \mid VP)$$
$$* \ p(VP \rightarrow flies \mid VP) * \ldots$$

Already have a CKY alg for weights ...



w(| s) = w(s ì NP VP) + w(NP ì time)

+ w(VP ì VP NP)

+ w(VP ì flies) + ...

Just let w(x ì Y Z) = -log p(x ì Y Z | x)

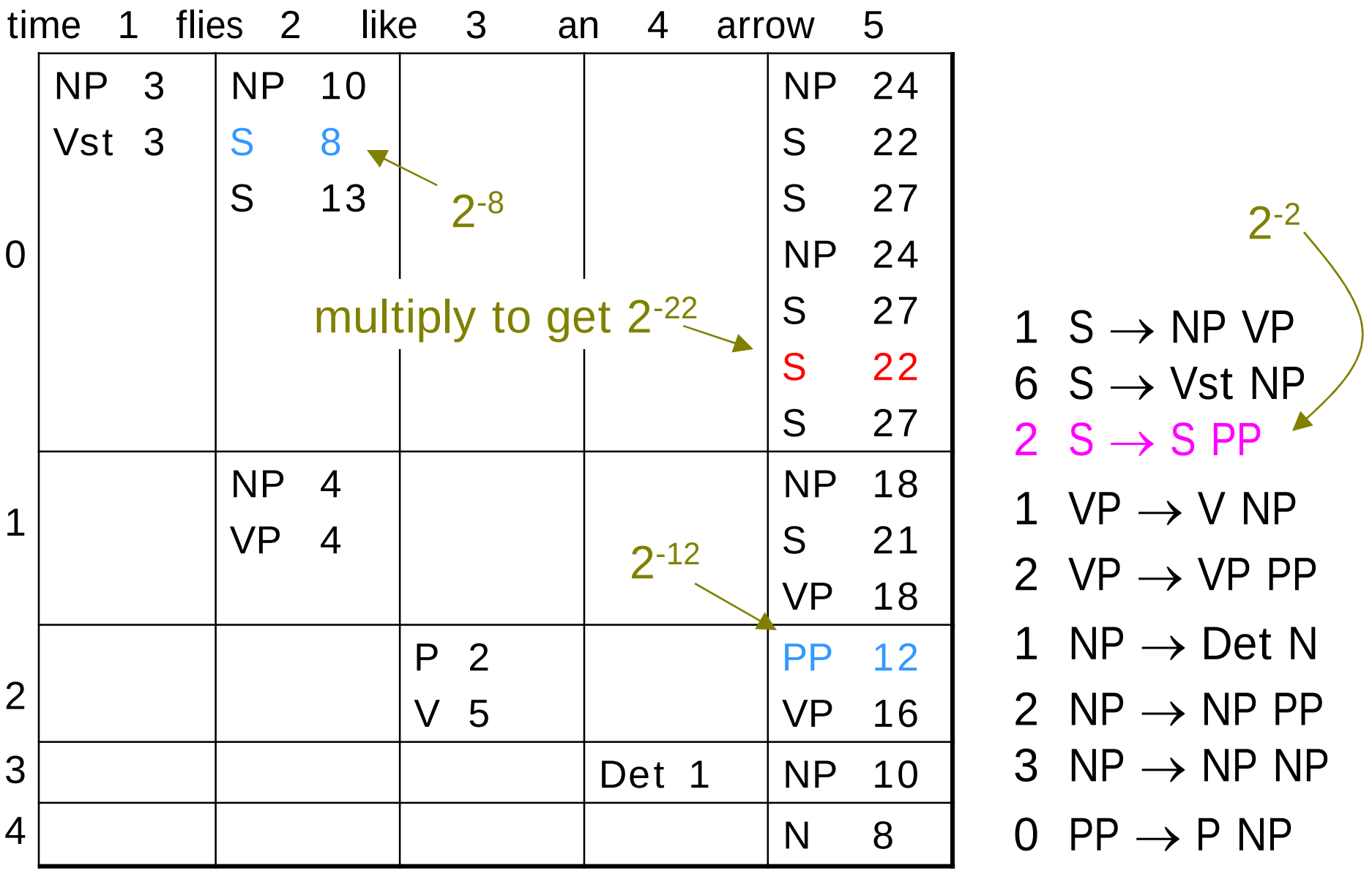
Then lightest tree has highest prob

Weighted Chart Parsing ("min cost")

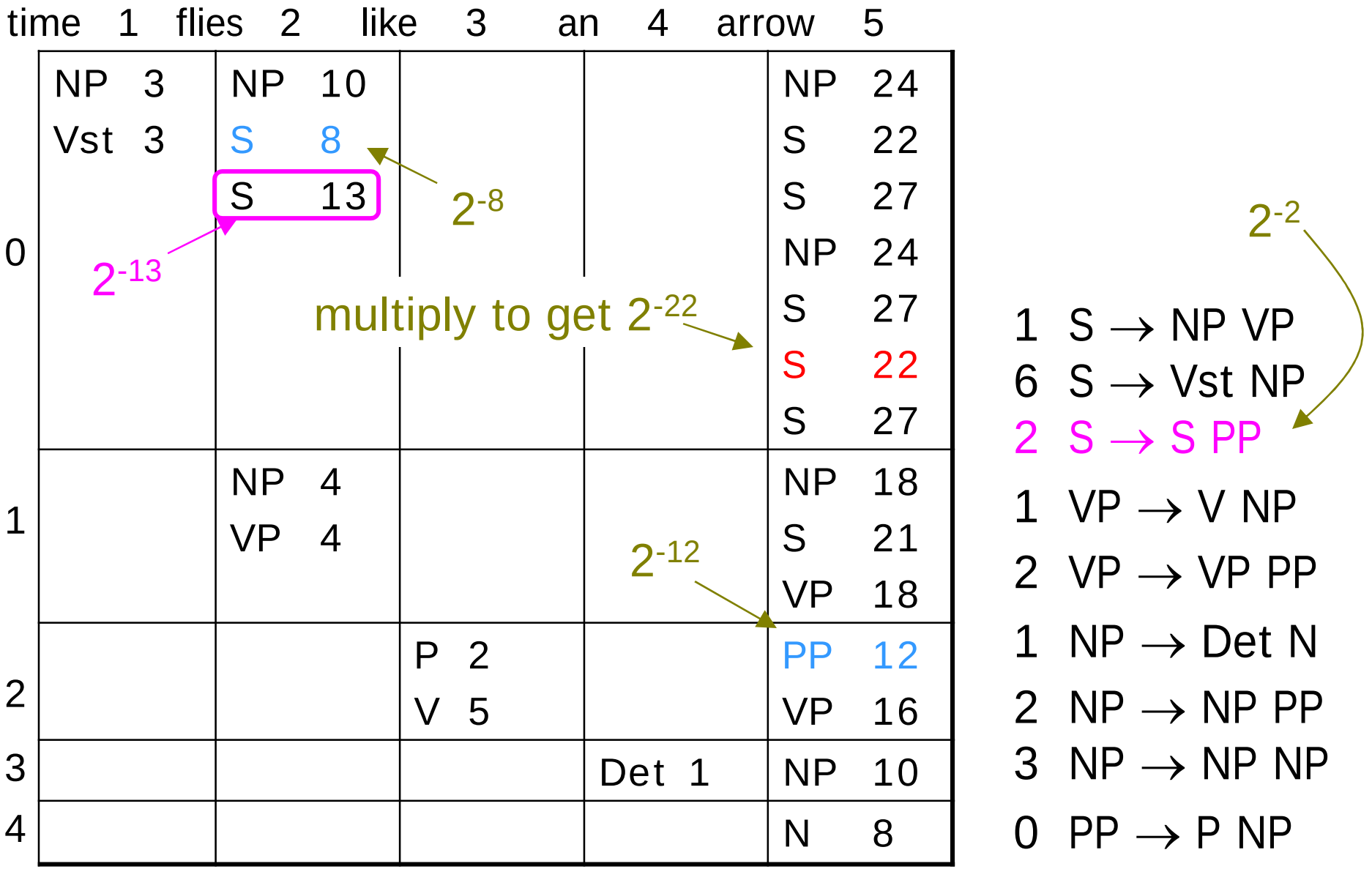
```
phrase(X,I,J)  min= rewrite(X,W) + word(W,I,J).
phrase(X,I,J)  min= rewrite(X,Y,Z) + phrase(Y,I,Mid) + phrase(Z,Mid,J).
goal           min=  phrase(start_symbol, 0, sentence_length).
```

Probabilistic Chart Parsing ("max prob")

```
phrase(X,I,J)  max= rewrite(X,W) * word(W,I,J).
phrase(X,I,J)  max= rewrite(X,Y,Z) * phrase(Y,I,Mid) * phrase(Z,Mid,J).
goal           max= phrase(start_symbol, 0, sentence_length).
```


time 1 flies 2 like 3 an 4 arrow 5
0
NP 3
Vst 3
NP 10
S 8
S 13
2-8
NP 24
S 22
S 27
NP 24
S 27
S 22
S 27
multiply to get 2-22
2-2
1
NP 4
VP 4
NP 18
S 21
VP 18
2-12
2
P 2
V 5
PP 12
VP 16
3
Det 1
NP 10
4
N 8
1 S → NP VP
6 S → Vst NP
2 S → S PP
1 VP → V NP
2 VP → VP PP
1 NP → Det N
2 NP → NP PP
3 NP → NP NP
0 PP → P NP

Need only best-in-class to get best parse



Why probabilities not weights?

- We just saw probabilities are really just a special case of weights ...
- ... but we can estimate them from training data by counting and smoothing! Yay!
  - Warning: What kind of training corpus do we need (if we want to estimate rule probabilities simply by counting and smoothing)?
- Probabilities tell us how likely our best parse actually is:
  - Might improve user interface (e.g. ask for clarification if not sure)
  - Might help when learning the rule probabilities (later in course)
  - Should help combination with other systems
    - Text understanding: Even if the 3rd-best parse is 40x less probable syntactically, might still use it if it's> 40x more probable semantically
    - Ambiguity-preserving translation: If the top 3 parses are all probable, try to find a translation that would be ok regardless of which is correct

A slightly different task

- Been asking: What is probability of generating a given tree with your homework 1 generator?
  - To pick tree with highest prob: useful in parsing.

- But could also ask: What is probability of generating a given string with the generator?
  (i.e., with the –t option turned off)
  - To pick string with highest prob: useful in speech recognition, as substitute for an n-gram model.
    - ("Put the file in the folder" vs. "Put the file and the folder")
  - To get prob of generating string, must add up probabilities of all trees for the string ...

Could just add up the parse probabilities

	time 1	flies 2	like 3	an 4	arrow 5
0	NP 3 Vst 3	NP 10 S 8 S 13			NP 24 S 22 S 27 NP 24 S 27 S 22 S 27
1		NP 4 VP 4			NP 18 S 21 VP 18
2			P 2 V 5		PP 12 VP 16
3				Det 1	NP 10
4					N 8

2^{-22} 2^{-27} 2^{-27} 2^{-22} 2^{-27}

oops, back to finding exponentially many parses

1 S → NP VP
6 S → Vst NP
2 S → S PP
1 VP → V NP
2 VP → VP PP
1 NP → Det N
2 NP → NP PP
3 NP → NP NP
0 PP → P NP

Any more efficient way?

	time 1	flies 2	like 3	an 4	arrow 5
0	NP 3 Vst 3	NP 10 S 2^{-8} S 2^{-13}			NP 24 S 22 S 27 NP 24 S 27 S 2^{-22} S 2^{-27}
1		NP 4 VP 4			NP 18 S 21 VP 18
2			P 2 V 5		PP 2^{-12} VP 16
3				Det 1	NP 10
4					N 8

1 S → NP VP
6 S → Vst NP
2^{-2} S → S PP
1 VP → V NP
2 VP → VP PP
1 NP → Det N
2 NP → NP PP
3 NP → NP NP
0 PP → P NP

Add as we go ... (the "inside algorithm")

	time 1	flies 2	like 3	an 4	arrow 5
0	NP 3 Vst 3	NP 10 S $2^{-8}+2^{-13}$			NP 24 S 22 S 27 NP 24 S 27 S 2^{-22} $+2^{-27}$
1		NP 4 VP 4			NP 18 S 21 VP 18
2			P 2 V 5		PP 2^{-12} VP 16
3				Det 1	NP 10
4					N 8

1 S → NP VP
6 S → Vst NP
2^{-2} S → S PP
1 VP → V NP
2 VP → VP PP
1 NP → Det N
2 NP → NP PP
3 NP → NP NP
0 PP → P NP

Add as we go ... (the "inside algorithm")

	time 1 flies 2	like 3	an 4	arrow 5
0	NP 3 Vst 3	NP 10 S $2^{-8}+2^{-13}$		NP 2^{-22} $+2^{-27}$ S 2^{-22} $+2^{-27}$ $+2^{-27}$ $+(2^{-22}$ $+2^{-27})$
1		NP 4 VP 4		NP 18 S 21 VP 18
2		P 2 V 5		PP 2^{-12} VP 16
3			Det 1	NP 10
4				N 8

1 S → NP VP
6 S → Vst NP
2^{-2} S → S PP
1 VP → V NP
2 VP → VP PP
1 NP → Det N
2 NP → NP PP
3 NP → NP NP
0 PP → P NP

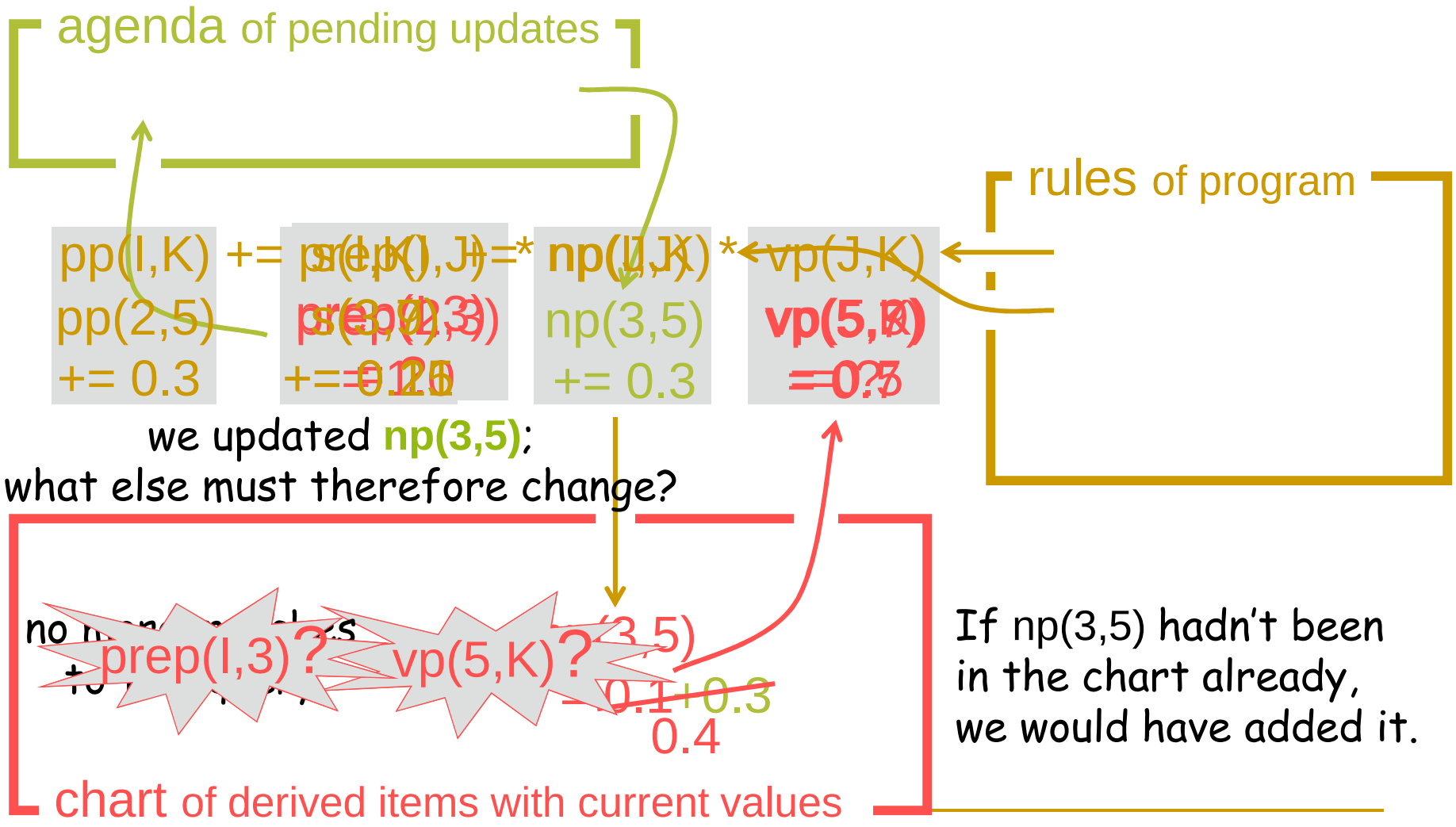
Probabilistic Chart Parsing ("max prob")

```
phrase(X,I,J)  max= rewrite(X,W) * word(W,I,J).
phrase(X,I,J)  max= rewrite(X,Y,Z) * phrase(Y,I,Mid) * phrase(Z,Mid,J).
goal           max= phrase(start_symbol, 0, sentence_length).
```

The "Inside Algorithm"

```
phrase(X,I,J)  +=  rewrite(X,W) * word(W,I,J).
phrase(X,I,J)  +=  rewrite(X,Y,Z) * phrase(Y,I,Mid) * phrase(Z,Mid,J).
goal           +=  phrase(start_symbol, 0, sentence_length).
```

Bottom-up inference



Parameterization …

```
phrase(X,I,J)   +=  rewrite(X,W)   * word(W,I,J).
phrase(X,I,J)   +=  rewrite(X,Y,Z) * phrase(Y,I,Mid) * phrase(Z,Mid,J).
goal            +=  phrase(start_symbol, 0, sentence_length).
```

- rewrite(X,Y,Z)'s value represents the rule probability p(Y Z | X).
- This could be defined by a formula instead of a number.
- Simple conditional log-linear model (each rule has 4 features):
  - urewrite(X,Y,Z) *= exp(weight_xy(X,Y)). % exp $\theta_{xy,X,y}$
  - urewrite(X,Y,Z) *= exp(weight_xz(X,Z)).
  - urewrite(X,Y,Z) *= exp(weight_yz(Y,Z)).
  - urewrite(Same,Same,Z) *= exp(weight_adjunction). % exp $\theta_{adjunction}$
  - urewrite(X) += urewrite(X,Y,Z). % normalizing constant
  - rewrite(X,Y,Z) = urewrite(X,Y,Z) / urewrite(X). % p(X → Y Z | X)

Parameterization …

```
phrase(X,I,J)   += rewrite(X,W)   * word(W,I,J).
phrase(X,I,J)   += rewrite(X,Y,Z) * phrase(Y,I,Mid) * phrase(Z,Mid,J).
goal            += phrase(start_symbol, 0, sentence_length).
```

- rewrite(X,Y,Z)'s value represents the rule probability p(Y Z | X).
- Simple conditional log-linear model …

- What if the program uses the unnormalized probability urewrite instead of rewrite?
- Different model! Each parse has an overall unnormalized prob:
  uprob(Parse) = exp(total weight of all features in the parse)
- Can still normalize at the end:
  p(Parse | sentence) = uprob(parse) / Z
  where Z is the sum of all uprob(Parse): given by goal!

Chart Parsing: Recognition algorithm

```
phrase(X,I,J) :- rewrite(X,W), word(W,I,J).
phrase(X,I,J) :- rewrite(X,Y,Z), phrase(Y,I,Mid), phrase(Z,Mid,J).
goal          :- phrase(start_symbol, 0, sentence_length).
```

Chart Parsing: Viterbi algorithm (min-cost)

```
phrase(X,I,J)  min= rewrite(X,W) + word(W,I,J).
phrase(X,I,J)  min= rewrite(X,Y,Z) + phrase(Y,I,Mid) + phrase(Z,Mid,J).
goal           min=  phrase(start_symbol, 0, sentence_length).
```

Chart Parsing: Viterbi algorithm (max-prob)

```
phrase(X,I,J)  max= rewrite(X,W) * word(W,I,J).
phrase(X,I,J)  max= rewrite(X,Y,Z) * phrase(Y,I,Mid) * phrase(Z,Mid,J).
goal           max= phrase(start_symbol, 0, sentence_length).
```

Chart Parsing: Inside algorithm

```
phrase(X,I,J)  +=  rewrite(X,W) * word(W,I,J).
phrase(X,I,J)  +=  rewrite(X,Y,Z) * phrase(Y,I,Mid) * phrase(Z,Mid,J).
goal           +=  phrase(start_symbol, 0, sentence_length).
```

Generalization: Semiring-Weighted Chart Parsing

phrase(X,I,J) ă= rewrite(X,W) Ă word(W,I,J).

phrase(X,I,J) ă= rewrite(X,Y,Z) Ă phrase(Y,I,Mid) Ă phrase(Z,Mid,J).

goal ă= phrase(start_symbol, 0, sentence_length).

Unweighted CKY: Recognition algorithm

- initialize all entries of chart to false
- for i := 1 to n
  - for each rule R of the form X → word[i]
    - chart[X,i-1,i] | | = in_grammar(R)
- for width := 2 to n
  - for start := 0 to n-width
    - Define end := start + width
    - for mid := start+1 to end-1
      - for each rule R of the form X → Y Z
      - chart[X,start,end] | | = in_grammar(R) && chart[Y,start,mid] && chart[Z,mid,end]
- return chart[ROOT,0,n]

Pay attention to the orange code …

Weighted CKY: Viterbi algorithm (min-cost)

- initialize all entries of chart to ã
- for i := 1 to n
  - for each rule R of the form X → word[i]
    - chart[X,i-1,i] min= weight(R)
- for width := 2 to n
  - for start := 0 to n-width
    - Define end := start + width
    - for mid := start+1 to end-1
      - for each rule R of the form X → Y Z
      - chart[X,start,end] min= weight(R) + chart[Y,start,mid] + chart[Z,mid,end]
- return chart[ROOT,0,n]

Pay attention to the orange code …

Probabilistic CKY: Inside algorithm

- initialize all entries of chart to 0
- for i := 1 to n
  - for each rule R of the form X → word[i]
    - chart[X,i-1,i] += prob(R)
- for width := 2 to n
  - for start := 0 to n-width
    - Define end := start + width
    - for mid := start+1 to end-1
      - for each rule R of the form X → Y Z
      - chart[X,start,end] += prob(R) * chart[Y,start,mid] * chart[Z,mid,end]
- return chart[ROOT,0,n]

Pay attention to the orange code ...

Semiring-weighted CKY: General algorithm!

- initialize all entries of chart to ⓪
- for i := 1 to n
  - for each rule R of the form X → word[i]
    - chart[X,i-1,i] ă= semiring_weight(R)
- for width := 2 to n
  - for start := 0 to n-width
    - Define end := start + width
    - for mid := start+1 to end-1
      - for each rule R of the form X → Y Z
      - chart[X,start,end] ă= semiring_weight(R) Ă chart[Y,start,mid] Ă chart[Z,mid,end]
- return chart[ROOT,0,n]

Ă is like "and"/∀: combines all of several pieces into an X

ă is like "or"/∃: considers the alternative ways to build the X

Semiring-weighted CKY: General algorithm!

- initialize all entries of chart to ⓪
- for i := 1 to n
  - for each rule R of the form X → word[i]
    - chart[X,i-1,i] ⊕= semiring_weight(R)
- for width := 2 to n
  - for start := 0 to n-width
    - Define end := start + width
    - for mid := start+1 to end-1
      - for each rule R of the form X → Y Z
      - chart[X,start,end] ⊕= semiring_weight(R) ⊗ chart[Y,start,mid] ⊗ chart[Z,mid,end]
- return chart[ROOT,0,n]

?

Weighted CKY, general version

- initialize all entries of chart to ⓪
- for i := 1 to n
  - for each rule R of the form X → word[i]
    - chart[X,i-1,i] ⊕= semiring_weight(R)
  - for width := 2 to n
    - for start := 0 to n-width
      - end := start + width
      - for mid := start+1 to end-1
        - for each rule R of the form X → Y Z
          - chart[X,start,end] ⊕= semiring_weight(R) ⊗ chart[Y,start,mid] ⊗ chart[Z,mid,end]
- return chart[ROOT,0,n]

	weights	⊕	⊗	⓪
total prob (inside)	[0,1]	+	×	0
min weight	[0,∞]	min	+	∞
recognizer	{true,false}	or	and	false

Other Uses of Semirings

- The semiring weight of a constituent, Chart[X,i,k], is a flexible bookkeeping device.
- If you want to build up information about larger constituents from smaller ones, design a semiring:
  - Probability of best parse, or its log
  - Number of parses
  - Total probability of all parses, or its log
  - The gradient of the total log-probability with respect to the parameters (use this to optimize the grammar weights)
  - The entropy of the probability distribution over parses
  - The 5 most probable parses
  - Possible translations of the constituent (this is how MT is done!)
- We'll see semirings again later with finite-state machines.

Some Weight Semirings

	weights	⊕	⊗	⓪	①
total prob (inside)	[0,1]	+	×	0	1
max prob	[0,1]	max	×	0	1
min weight = -log(max prob)	[0,∞]	min	+	∞	0
log(total prob)	[-∞,0]	log+	+	-∞	0
recognizer	{true,false}	or	and	false	true

log

Semiring elements are log-probabilities lp, lq; helps prevent underflow

lp ⊗ lq = log(exp(lp)×exp(lq)) = lp+lq

lp ⊕ lq = log(exp(lp)+exp(lq)), denoted log+(lp,lq)

The Semiring Interface

```
public interface Semiring<K extends Semiring> {
   public K oplus(K k);   // ⊕
   public K otimes(K k);  // ⊗
   public K zero();       // ⓪
   public K one();        // ①
}
public class Minweight implements Semiring<Minweight> {
   protected float f;
   public Minweight(float f) { this.f =  f; }
   public float toFloat() { return f; }
   public Minweight oplus(Minweight k) {
      return (f <= k.toFloat()) ? this : k; }
   public Minweight otimes(Minweight k) {
      return new Minweight(f + k.toFloat()); }
   static Minweight ZERO = new Minweight(Float.POSITIVE_INFINITY);
   static Minweight ONE = new Minweight(0);
   public Minweight zero() { return ZERO; }
   public Minweight one() { return ONE; } }
```

A Generic Parser for Any Semiring K

```
public interface Semiring<K extends Semiring> {
   public K oplus(K k);   // ⊕
   public K otimes(K k);  // ⊗
   public K zero();       // ⓪
   public K one();        // ①
}

public class ContextFreeGrammar<K extends Semiring<K>> {
   … // CFG with rule weights in K
}
public class CKYParser<K extends Semiring<K>> {
   … // parser for a CFG whose rule weights are in K
   K parse(Vector<String> input) { … }
      // returns “total” weight (using ⊕) of all parses
}
g = new ContextFreeGrammar<Minweight>(…);
p = new CKYParser<Minweight>(g);
minweight = p.parse(input);  // returns min weight of any parse
```

The Semiring Axioms

```
public interface Semiring<K extends Semiring> {
  public K oplus(K k);   // ⊕
  public K otimes(K k);  // ⊗
  public K zero();       // 0
  public K one();        // 1
}
```

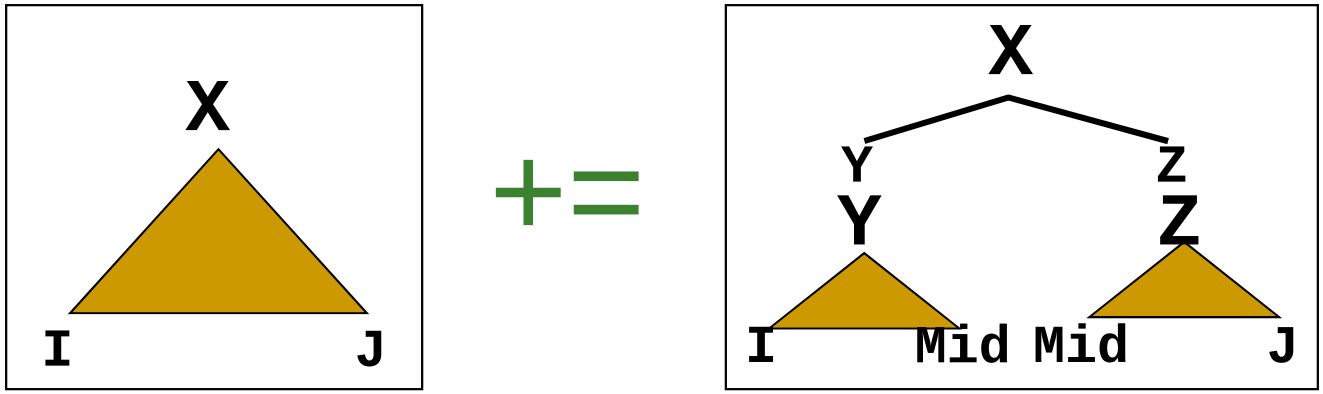
An implementation of **Semiring** must satisfy the semiring axioms:

- Commutativity of ⊕: $a \oplus b = b \oplus a$
- Associativity: $(a \oplus b) \oplus c = a \oplus (b \oplus c)$,
  $(a \otimes b) \otimes c = a \otimes (b \otimes c)$
- Distributivity: $a \otimes (b \oplus c) = (a \otimes b) \oplus (a \otimes c)$,
  $(b \oplus c) \otimes a = (b \otimes a) \oplus (c \otimes a)$
- Identities: $a \oplus 0 = 0 \oplus a = a$, $a \otimes 1 = 1 \otimes a = a$
- Annihilation: $a \otimes 0 = 0$

Otherwise the parser won't work correctly. Why not? (Look back at it.)

Rule binarization can speed up program

phrase(X,I,J) += rewrite(X,Y,Z) * phrase(Y,I,Mid) * phrase(Z,Mid,J).

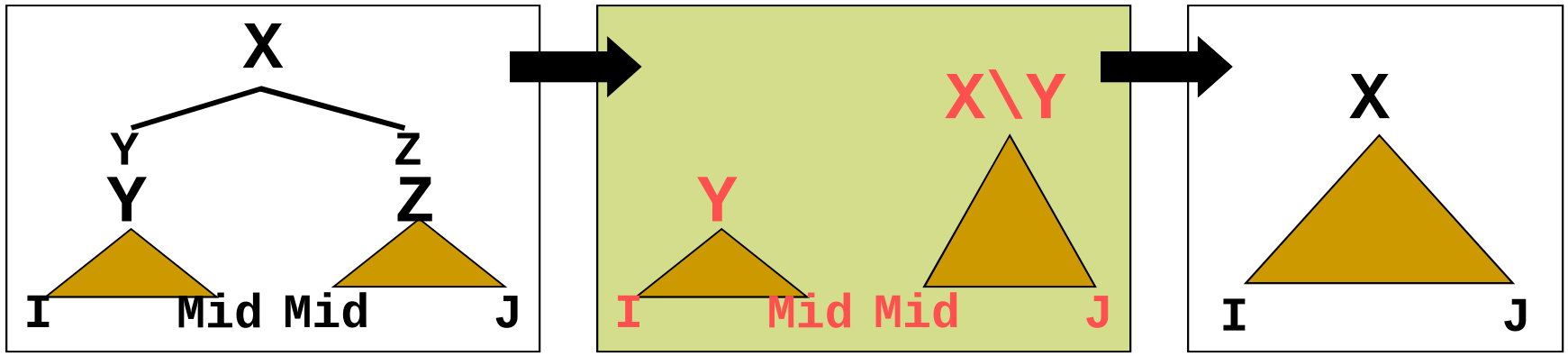


Rule binarization can speed up program

```
phrase(X,I,J)    +=  phrase(Y,I,Mid) * phrase(Z,Mid,J) * rewrite(X,Y,Z).
```

folding transformation: asymp. speedup!

```
temp(X\Y,Mid,J)  +=  phrase(Z,Mid,J) * rewrite(X,Y,Z).
phrase(X,I,J)    +=  phrase(Y,I,Mid) * temp(X\Y,Mid,J).
```



Rule binarization can speed up program

phrase(X,I,J) += phrase(Y,I,Mid) * phrase(Z,Mid,J) * rewrite(X,Y,Z).

folding transformation: asymp. speedup!

temp(X\Y,Mid,J) += phrase(Z,Mid,J) * rewrite(X,Y,Z).

phrase(X,I,J) += phrase(Y,I,Mid) * **temp**(X\Y,Mid,J).

$$\sum_{Y,Z,Mid} \text{phrase(Y,I,Mid)} * \text{phrase(Z,Mid,J)} * \text{rewrite(X,Y,Z)}$$

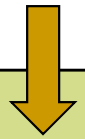
graphical models
constraint programming
multi-way database join

$$\sum_{Y,Mid} \text{phrase(Y,I,Mid)} \sum_{Z} \text{phrase(Z,Mid,J)} * \text{rewrite(X,Y,Z)}$$

Earley's algorithm in Dyna

```
phrase(X,I,J)   += rewrite(X,W)   * word(W,I,J).
phrase(X,I,J)   += rewrite(X,Y,Z) * phrase(Y,I,Mid) * phrase(Z,Mid,J).
goal            += phrase(start_symbol, 0, sentence_length).
```

magic templates transformation (as noted by Minnen 1996)



```
need(start_symbol,0) = true.
need(Nonterm,J) :- phrase(_/[Nonterm|_],_,J).
phrase(Nonterm/Needed,I,I)
                += need(Nonterm,I), rewrite(Nonterm,Needed).
phrase(Nonterm/Needed,I,K)
                += phrase(Nonterm/[W|Needed],I,J) * word(W,J,K).
phrase(Nonterm/Needed,I,K)
                += phrase(Nonterm/[X|Needed],I,J) * phrase(X/[],J,K).
goal += phrase(start_symbol/[],0,sentence_length).
```