تمرین سوم درس پردازش زبان طبیعی

نام تهیه‌کننده: ملیکا نوبختیان

نسخه: ۱

۱- بخش تئوری

۱-۱- سوال اول

تعداد Verb = ۴ تعداد Modal = ۴ تعداد Noun = ۶

Emission probability

کلمات	Noun	Modal	Verb
Mark	1/6	0	2/4
Can	0	3/4	0
Watch	2/6	0	2/4
Will	1/6	1/4	0
Tom	2/6	0	0

Sentences:

<S> Mark can watch. <E>
N M V

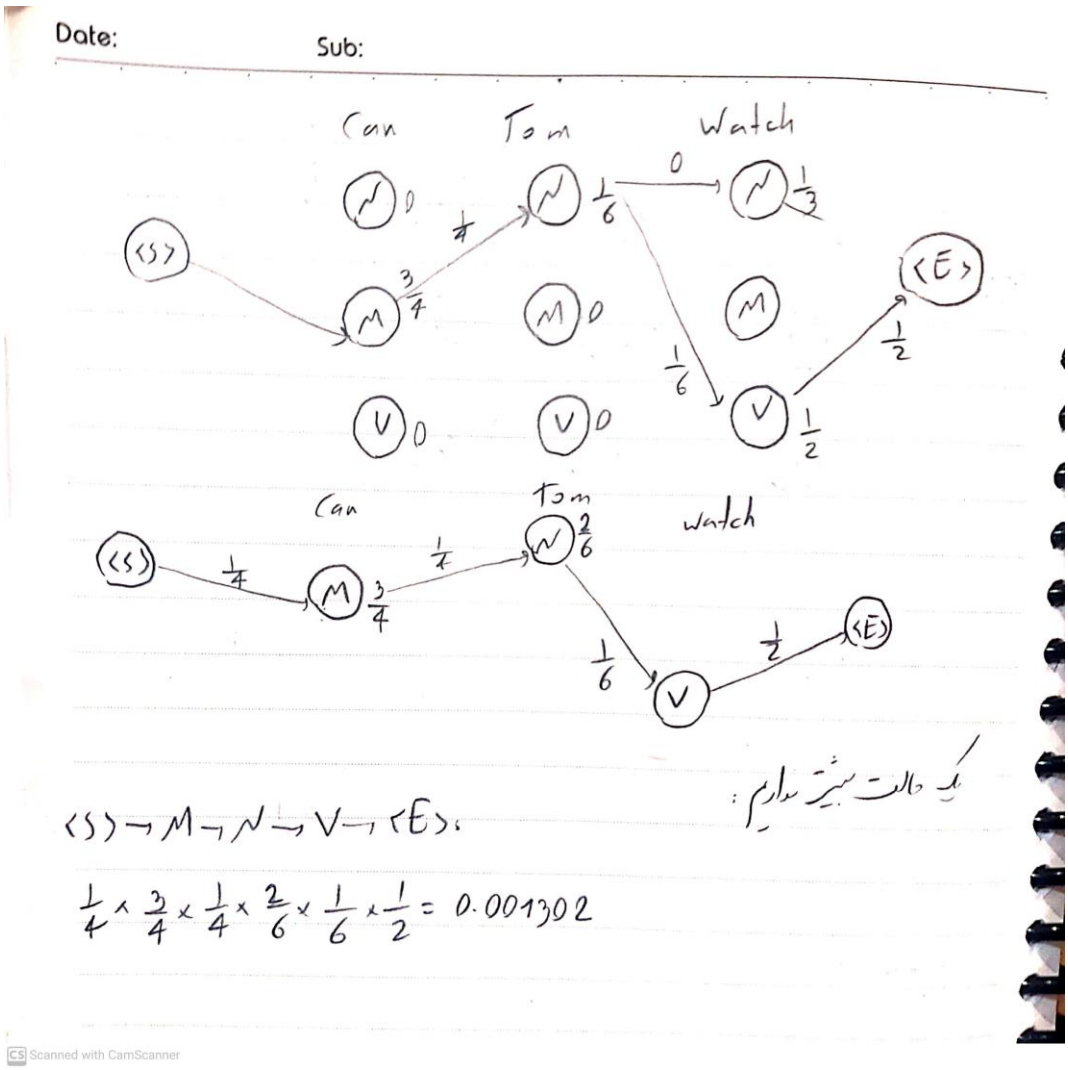
<S> Will can mark watch. <E>
N M V N

<S> Can Tom watch? <E>
M N V

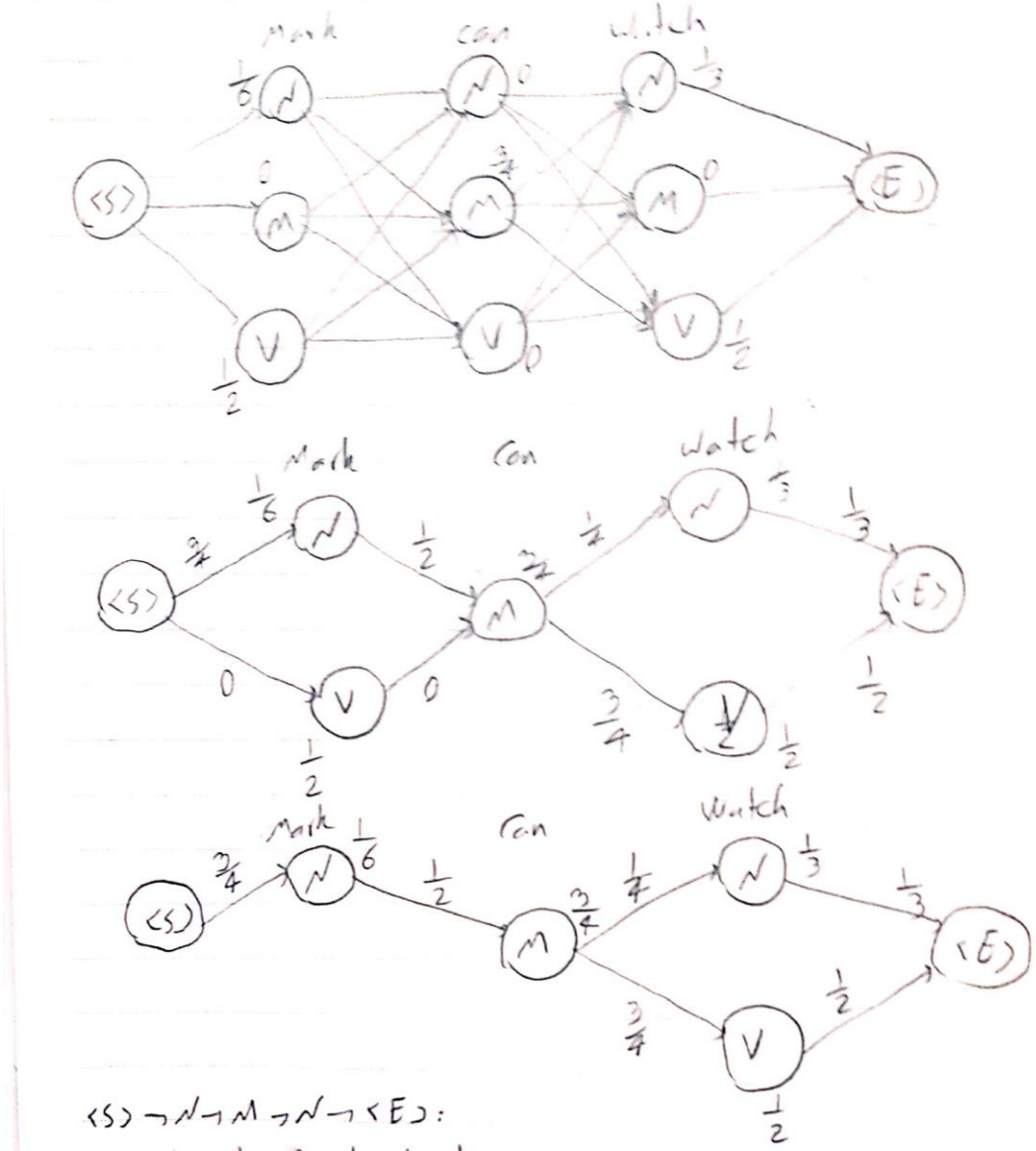
<S> Tom will mark watch. <E>
N M V N

Transition Probablity:

	Noun	Modal	Verb	<E>
<S>	3/4	1/4	0	0
Noun	0	3/6	1/6	2/6
Modal	1/4	0	3/4	0
Verb	2/4	0	0	2/4



Date: Sub:



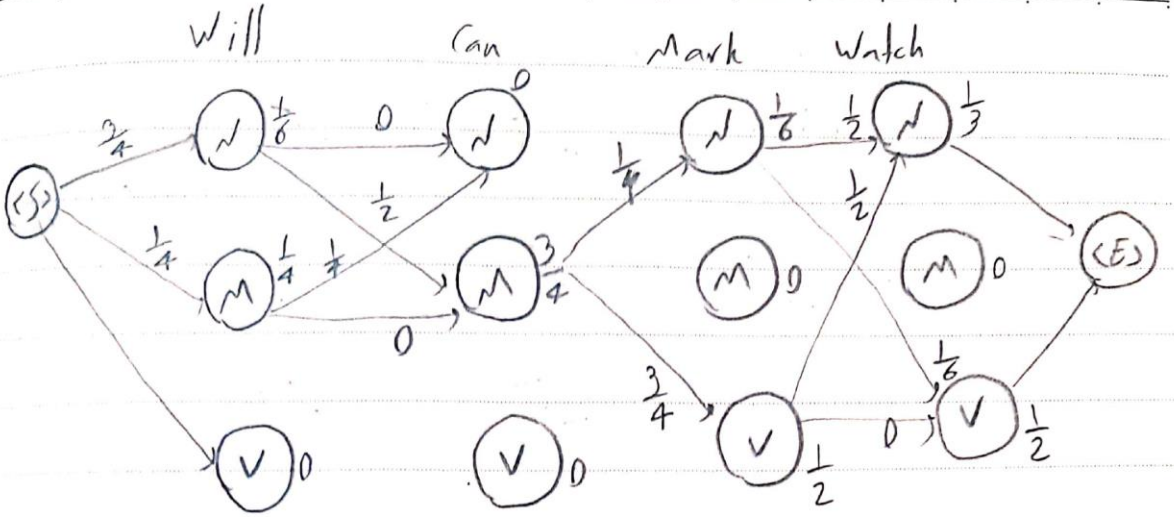
$\langle S\rangle \to N \to M \to N \to \langle E\rangle$:

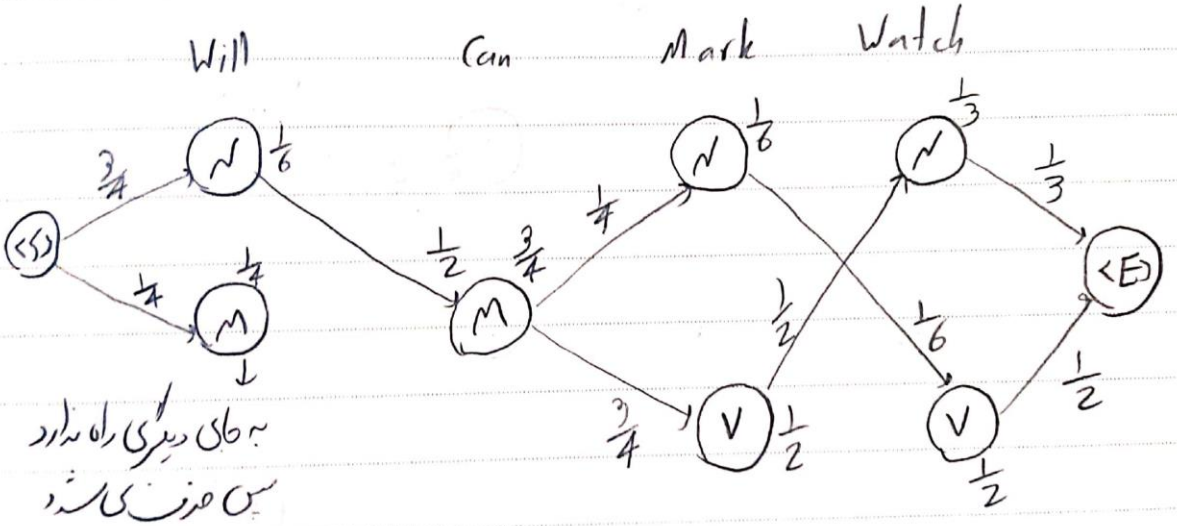
$\frac{3}{4} \times \frac{1}{6} \times \frac{1}{2} \times \frac{3}{4} \times \frac{1}{4} \times \frac{1}{3} \times \frac{1}{3} = 0.001302$

$\langle S\rangle \to N \to M \to V \to \langle E\rangle$:

$\frac{3}{4} \times \frac{1}{6} \times \frac{1}{2} \times \frac{3}{4} \times \frac{3}{4} \times \frac{1}{2} \times \frac{1}{2} = 0.00878$ ✓

Date: Sub:





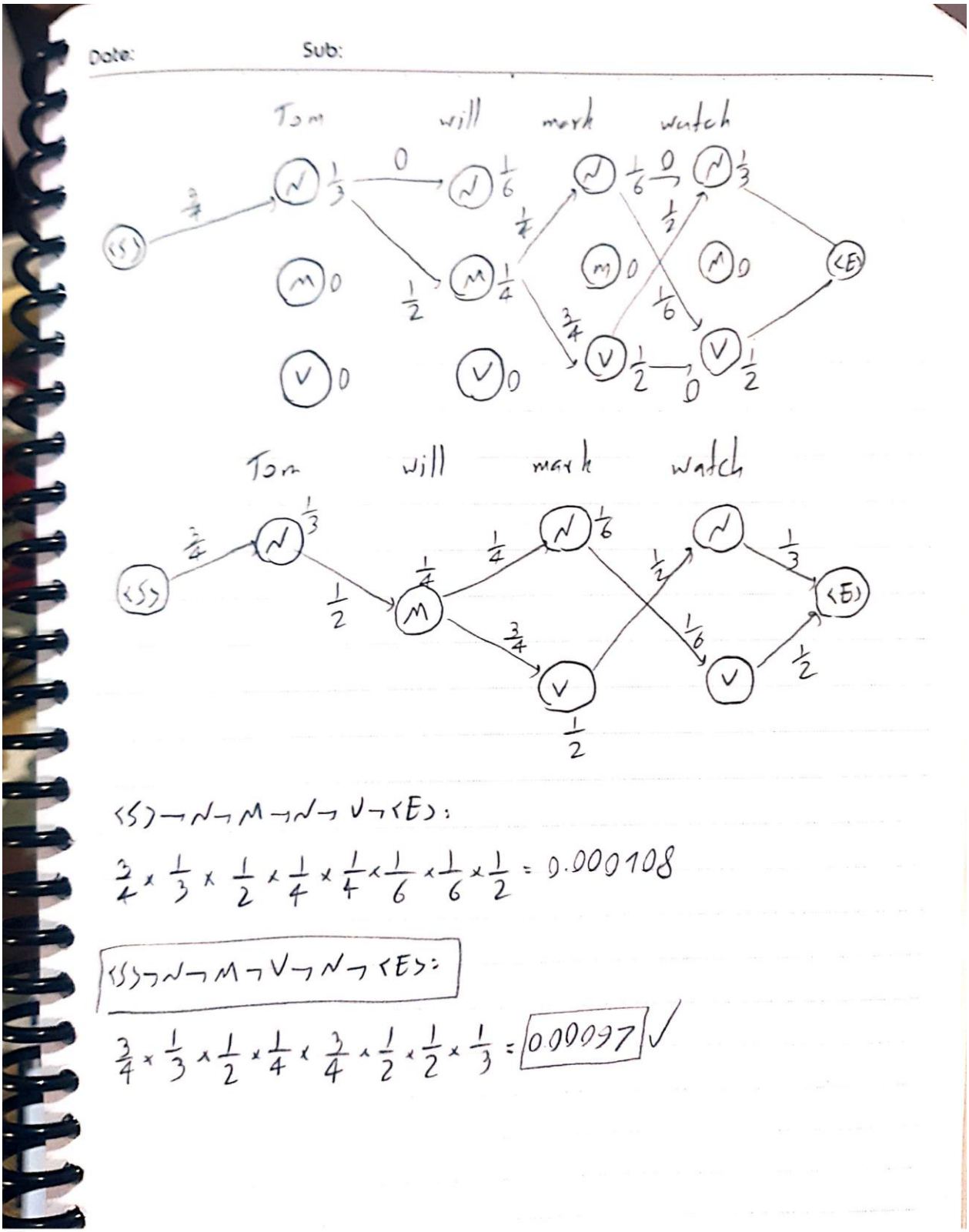
جای دیگری راه ندارد
پس حذف می‌شود

$\langle S\rangle \to N \to M \to N \to V \to \langle E\rangle$:

$\frac{3}{4}\times\frac{1}{6}\times\frac{1}{2}\times\frac{3}{4}\times\frac{1}{4}\times\frac{1}{6}\times\frac{1}{6}\times\frac{1}{2}\times\frac{1}{2} = 0.0000813$

$\langle S\rangle \to N \to M \to V \to N \to \langle E\rangle$: ✓

$\frac{3}{4}\times\frac{1}{6}\times\frac{1}{2}\times\frac{3}{4}\times\frac{3}{4}\times\frac{1}{2}\times\frac{1}{2}\times\frac{1}{3}\times\frac{1}{3} = 0.000976$ ✓



$\langle S\rangle \to N \to M \to N \to V \to \langle E\rangle$:

$\frac{3}{4} \times \frac{1}{3} \times \frac{1}{2} \times \frac{1}{4} \times \frac{1}{4} \times \frac{1}{6} \times \frac{1}{6} \times \frac{1}{2} = 0.000108$

$\langle S\rangle \to N \to M \to V \to N \to \langle E\rangle$:

$\frac{3}{4} \times \frac{1}{3} \times \frac{1}{2} \times \frac{1}{4} \times \frac{3}{4} \times \frac{1}{2} \times \frac{1}{2} \times \frac{1}{3} = 0.00097$ ✓

۲-۱- سوال دوم

Date: Sub:

(۲)

ابتدا باید گرامر را برای ساختن جدول به نرمال چامسکی تبدیل کنیم:

$S' \rightarrow S$ 1

$S \rightarrow NP\ VP$

$S \rightarrow SC\ S$ 0.2 $SC \rightarrow S\ conj$ 1

$NP \rightarrow Noun \mid Det\ Noun \mid NP\ PP \mid NPC\ NP$ 0.2

$NPC \rightarrow NP\ conj$ 1

$VP \rightarrow Verb \mid Verb\ NP \mid Verb\ 2NP \mid VP\ PP$ 0.1

$2NP \rightarrow NP\ NP$ 1

$PP \rightarrow P\ NP$

$Noun \rightarrow John \mid Jack \mid pie \mid cream \mid cake$

$Verb \rightarrow eat \mid eats \mid drinks$ $P \rightarrow with \mid by$

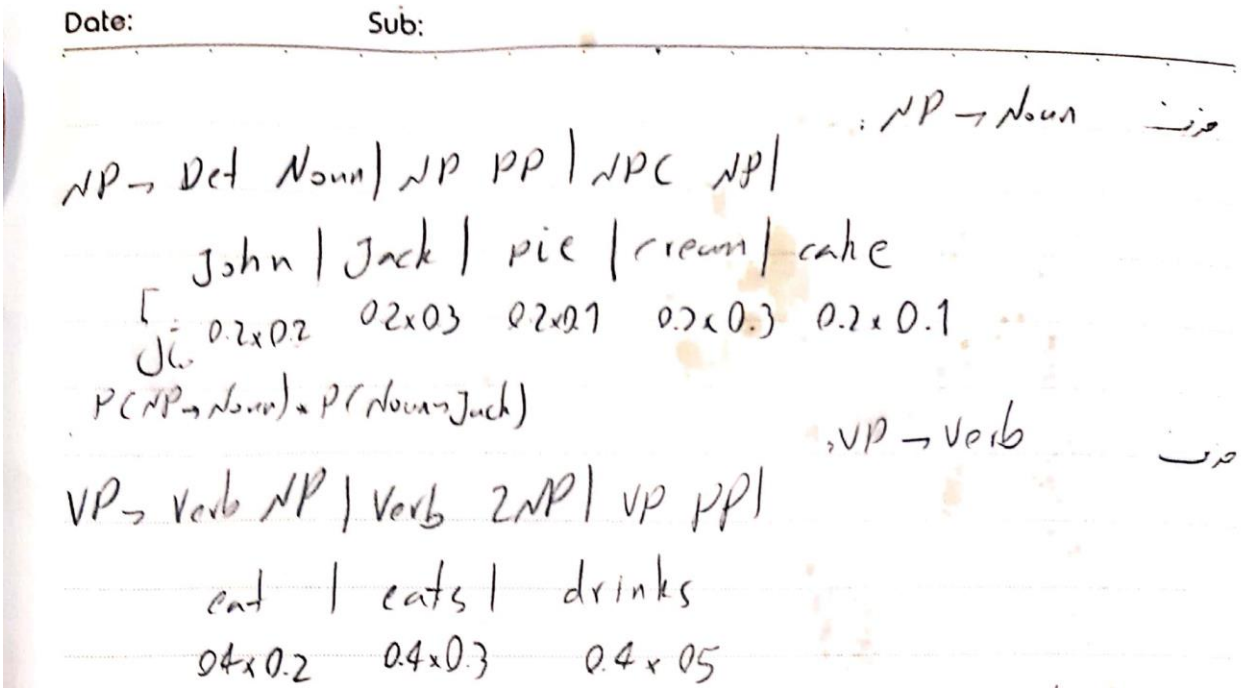
$Det \rightarrow a \mid the$ $conj \rightarrow and \mid or$

حذف $S' \rightarrow S$:

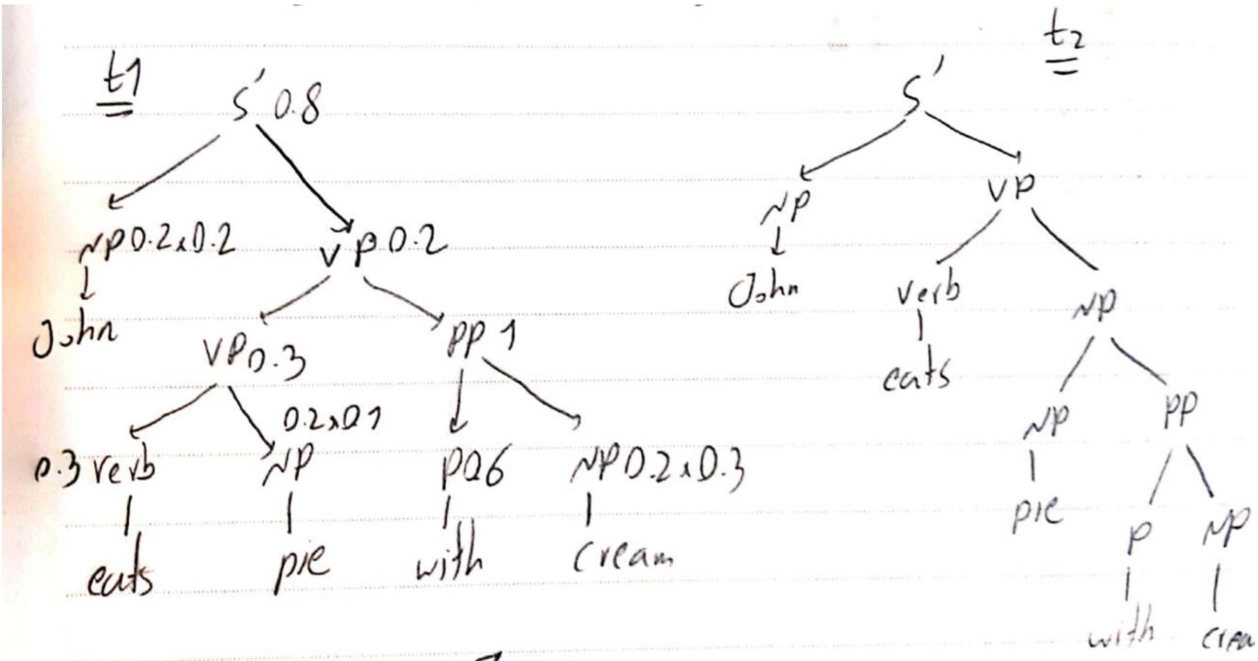
$S' \rightarrow NP\ VP \mid SC\ S$

1×0.8 1×0.2

Elipon



John	**eats**	**pie**	**with**	**cream**	
NP, Noun	**S', S**	**S',S**	-	**S',S**	**john**
	VP, Verb	**VP**	-	**VP**	**eats**
		NP, Noun	-	**NP**	**pie**
			P	**PP**	**with**
				NP, Noun	**cream**



$P(t1) = 4/1472 \times 10^{-7}$

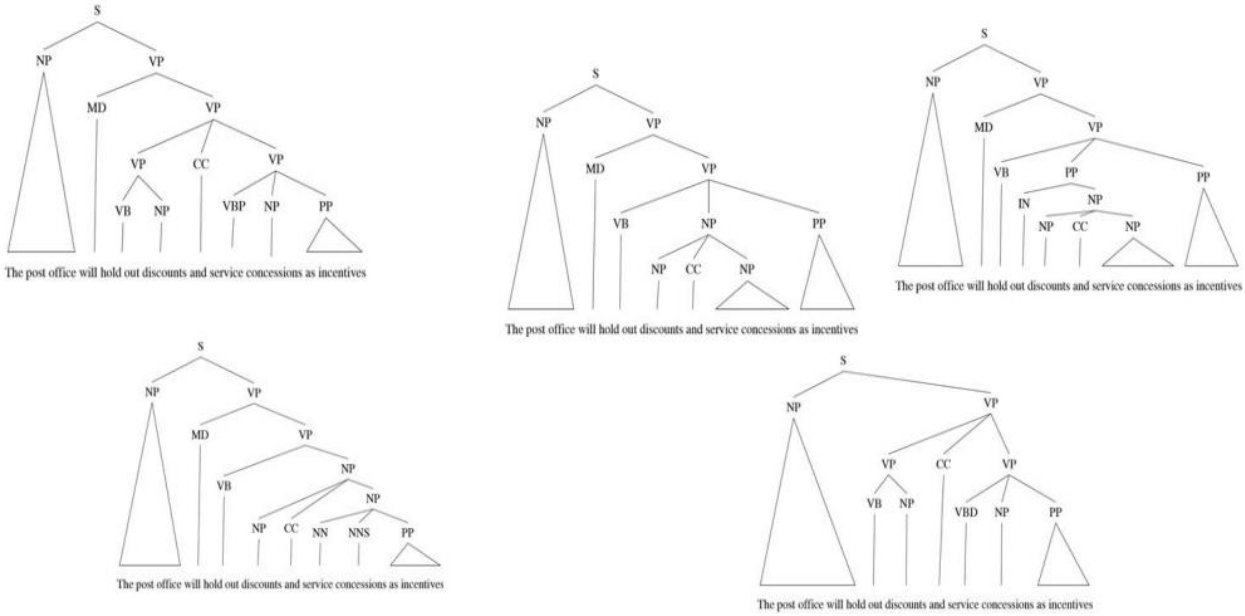
$P(t2) = 4/1472 \times 10^{-7}$

احتمال برابر دارند پس هر دو حالت می‌توانند انتخاب بشوند

۳-۱- سوال سوم

اولین مشکل PCFG ها این است اهمیتی به lexical rules یا قواعد لغوی قائل نمی‌شوند به این دلیل ممکن است صرفا به دلیل اینکه یک rule احتمال بالاتری دارد آن ساختار انتخاب خواهد شد بدون توجه به اینکه نگاهی به قواعد لغوی داشته باشد در حالی که این موارد مهم هستند و ساختار اصلی را مشخص می کنند. هم چنین ممکن است قواعدی که در حالت‌های مختلف parse کردن یک جمله استفاده می‌شوند متفاوت باشند ولی در نهایت احتمال هر دو جمله یکسان شوند چون احتمال ها یکی بوده‌اند. تصویر زیر حالت‌های مختلف parse کردن یک جمله ر ابه ما نشان می‌دهد:

The post office will hold out discounts and service concessions as incentives.



با در نظر گرفتن lexicalized گرامرها این مشکل تا حدودی حل می‌شود چون در این گرامرها هر rule بر اساس کلمه‌ای که در حال حاضر داریم انتخاب می‌شود به این صورت که شاید احتمال یک rule با توجه به آن کلمه بیشتر یا کمتر باشد که در این صورت اطلاعات لغوی نیز به کار برده می‌شوند و پیش‌بینی دقیق‌تری خواهیم داشت. تصویر زیر احتمال یک rule با توجه به کلمات مختلف را نشان می‌دهد:

- $p(VP \rightarrow V\ NP\ NP)$ = 0.00151
- $p(VP \rightarrow V\ NP\ NP \mid said)$ = 0.00001
- $p(VP \rightarrow V\ NP\ NP \mid gave)$ = 0.01980 ”

مشکل دیگر PCFG ها bias داشتن به سمت درخت‌های کوتاه‌تر است و احتمال بیشتری را برای آن‌ها قائل می‌شود در حالی که ممکن است چنین فرضی درست نباشد. Lexicalized گرامرها تغییر چندانی در این وضع ایجاد نمی‌کنند ولی شاید اطلاعات لغوی وضعیت را کمی بهتر کند.

۲- سوال عملی

برای اینکه بتوانیم تصمیم گیری درستی در درخت تصمیم برای تشخیص tag ها داشته باشیم، لازم است ابتدا چند feature مناسب برای هر کلمه در جمله به دست آوریم تا بر اساس آن‌ها تصمیم بگیریم که چه tag را به آن کلمه نسبت دهیم. تابع features این کار را انجام می‌دهد. ویژگی‌های مختلفی در این تابع استخراج می‌شوند که از جمله آن‌ها می‌توان به پیشوند و پسوندهای کلمه و اینکه کلمه آخر یا اول جمله است و یا اینکه حروف بزرگ دارد یا نه اشاره کرد:

```
def features(sentence, index):
    return {
        'word': sentence[index],
        'is_first': index == 0,
        'is_last': index == len(sentence) - 1,
        'is_before_period': True if index != len(sentence) - 1 and sentence[index + 1] == '.' else False,
        'is_capitalized': sentence[index][0].upper() == sentence[index][0],
        'is_all_caps': sentence[index].upper() == sentence[index],
        'is_all_lower': sentence[index].lower() == sentence[index],
        'prefix-1': sentence[index][0],
        'prefix-2': sentence[index][:2],
        'suffix-1': sentence[index][-1],
        'suffix-2': sentence[index][-2:],
        'prev_word': '' if index == 0 else sentence[index - 1],
        'next_word': '' if index == len(sentence) - 1 else sentence[index + 1],
        'last_word_in_sentence': sentence[-1],
        'is_numeric': sentence[index].isdigit(),
        'capitals_inside': sentence[index][1:].lower() != sentence[index][1:]
    }
```

برای اینکه بتوانیم از جملات treebank برای آموزش استفاده کنیم لازم است tag های آن را حذف کنیم که از طریق تابع زیر این کار را انجام می دهیم:

```
def untag(tagged_sentence):
    return [w for w, t in tagged_sentence]
```

حالا باید داده‌ها را به دو بخش train وtest تقسیم کنیم که نسبت آن‌ها ۸۰ به ۲۰ است:

```
cutoff = int(.80 * len(tagged_sentences))
training_sentences = tagged_sentences[:cutoff]
test_sentences = tagged_sentences[cutoff:]
```

حالا لازم است که برای آموزش یک دیتاست بسازیم. برای این کار برای جمله‌های untag شده، feature های موردنیاز را استخراج می‌کنیم و tag کلمه در جمله نیز label آن خواهد بود:

```
def transform_to_dataset(tagged_sentences):
    X, y = [], []

    for tagged in tagged_sentences:
        for index in range(len(tagged)):
            X.append(features(untag(tagged), index))
            y.append(tagged[index][1])

    return X, y

X, y = transform_to_dataset(training_sentences)
```

حالا برای آموزش مدل یک پایپ لاین تعریف می‌کنیم. در این پایپ لاین در ابتدا یک vectorizer خواهیم داشت که ویژگی‌ها را به بردارهای مناسب نگاشت می‌کند تا بتوانیم با استفاده از آن‌ها مدل را آموزش دهیم. سپس این داده‌ها به classifier داده خواهند شد تا tag ها را پیش‌بینی کند. نتیجه روی داده‌های تست دقت ۸۹ درصدی را مطابق شکل زیر نشان می‌دهد:

```
clf = Pipeline([
    ('vectorizer', DictVectorizer(sparse=False)),
    ('classifier', DecisionTreeClassifier(criterion='entropy'))
])

clf.fit(X[:10000], y[:10000])

print ('Training completed')

X_test, y_test = transform_to_dataset(test_sentences)

print ("Accuracy:", clf.score(X_test, y_test))
```

```
Training completed
Accuracy: 0.8947552273067518
```

حالا با تابع pos_tag می‌توانیم هر جمله دلخواهی را tag کنیم:

```
def pos_tag(sentence):
  tags = clf.predict([features(sentence, index) for index in range(len(sentence))])
  return list(zip(sentence, tags))

pos_tag(word_tokenize('This is a test sentence!'))
```

```
[('This', 'DT'),
 ('is', 'VBZ'),
 ('a', 'DT'),
 ('test', 'NN'),
 ('sentence', 'NN'),
 ('!', 'CD')]
```