Solutions for Chapter 21
Reinforcement Learning

21.1 The code repository shows an example of this, implemented in the passive 4×3 environment. The agents are found under `lisp/learning/agents/passive*.lisp` and the environment is in `lisp/learning/domains/4x3-passive-mdp.lisp`. (The MDP is converted to a full-blown environment using the function `mdp->environment` which can be found in `lisp/uncertainty/environments/mdp.lisp`.)

21.2 Consider a world with two states, S_0 and S_1, with two actions in each state: stay still or move to the other state. Assume the move action is non-deterministic—it sometimes fails, leaving the agent in the same state. Furthermore, assume the agent starts in S_0 and that S_1 is a terminal state. If the agent tries several move actions and they all fail, the agent may conclude that $T(S_0, Move, S_1)$ is 0, and thus may choose a policy with $\pi(S_0) = Stay$, which is an improper policy. If we wait until the agent reaches S_1 before updating, we won't fall victim to this problem.

21.3 This question essentially asks for a reimplementation of a general scheme for asynchronous dynamic programming of which the prioritized sweeping algorithm is an example (Moore and Atkeson, 1993). For **a.**, there is code for a priority queue in both the Lisp and Python code repositories. So most of the work is the experimentation called for in **b**.

21.4 This utility estimation function is similar to equation (21.9), but adds a term to represent Euclidean distance on a grid. Using equation (21.10), the update equations are the same for θ_0 through θ_2, and the new parameter θ_3 can be calculated by taking the derivative with respect to θ_3:

$$\begin{aligned}
\theta_0 &\leftarrow \theta_0 + \alpha\,(u_j(s) - \hat{U}_\theta(s))\,, \\
\theta_1 &\leftarrow \theta_1 + \alpha\,(u_j(s) - \hat{U}_\theta(s))x\,, \\
\theta_2 &\leftarrow \theta_2 + \alpha\,(u_j(s) - \hat{U}_\theta(s))y\,, \\
\theta_3 &\leftarrow \theta_3 + \alpha\,(u_j(s) - \hat{U}_\theta(s))\sqrt{(x - x_g)^2 + (y - y_g)^2}\,.
\end{aligned}$$

21.5 Code not shown. Several reinforcement learning agents are given in the directory `lisp/learning/agents`.

21.6 Possible features include:

- Distance to the nearest $+1$ terminal state.

第21章强化学习的解决方案

21.1代码库显示了一个这样的例子，在被动的4×3环境中实现。 代理在lisp/学习/代理/被动*下找到。lisp和环境在lisp/learning/domains/4x3-passive-mdp。lisp。 （使用函数mdp->environment将MDP转换为完整的环境，该函数可以在lisp/不确定性/环境/mdp中找到。lisp。）

21.2考虑一个有两个状态的世界，S0和S1，每个状态有两个动作：保持静止或移动到另一个状态。 假设移动操作是非确定性的-它有时会失败，使代理处于相同的状态。 此外，假设代理从S0开始，并且S1是终端状态。 如果代理尝试多个移动操作并且它们都失败，则代理可能会得出t(S0,Move,S1)为0的结论，并且因此可能选择具有π(S0)=Stay的策略，这是不正确的策略。 如果我们在更新之前等到代理达到S1，我们就不会成为这个问题的牺牲品。

21.3这个问题基本上要求重新实现asyn chronous dynamic programming的一般方案，其中优先扫描算法是一个考试ple（Moore和Atkeson，1993）。 例如，Lisp和Python代码存储库中都有优先级队列的代码。 所以大部分工作都是b中要求的实验。

21.4该效用估计函数类似于等式（21.9），但添加了一个项来表示网格上的欧几里德距离。 使用方程(21.10)，更新方程对于θ0到θ2是相同的，并且可以通过取相对于θ3的导数来计算新参数θ3：

$$\begin{aligned}
\theta_0 &\leftarrow \theta_0 + \alpha\,(u_j(s) - \hat{U}_\theta(s))\,, \\
\theta_1 &\leftarrow \theta_1 + \alpha\,(u_j(s) - \hat{U}_\theta(s))x\,, \\
\theta_2 &\leftarrow \theta_2 + \alpha\,(u_j(s) - \hat{U}_\theta(s))y\,, \\
\theta_3 &\leftarrow \theta_3 + \alpha\,(u_j(s) - \hat{U}_\theta(s))\sqrt{(x - x_g)^2 + (y - y_g)^2}\,.
\end{aligned}$$

21.5未显示的代码。 在目录lisp/learning/agents中给出了几个强化学习代理。

21.6 可能的功能包括:

*到最近的+1终端状态的距离。

- Distance to the nearest -1 terminal state.
- Number of adjacent $+1$ terminal states.
- Number of adjacent -1 terminal states.
- Number of adjacent obstacles.
- Number of obstacles that intersect with a path to the nearest $+1$ terminal state.

21.7 The modification involves combining elements of the environment converter for games (`game->environment` in `lisp/search/games.lisp`) with elements of the function `mdp->environment`. The reward signal is just the utility of winning/drawing/losing and occurs only at the end of the game. The evaluation function used by each agent is the utility function it learns through the TD process. It is important to keep the TD learning process (which is entirely independent of the fact that a game is being played) distinct from the game-playing algorithm. Using the evaluation function with a deep search is probably better because it will help the agents to focus on relevant portions of the search space by improving the quality of play. There is, however, a tradeoff: the deeper the search, the more computer time is used in playing each training game.

21.8 This is a relatively time-consuming exercise. Code not shown to compute three-dimensional plots. The utility functions are:

a. $U(x, y) = 1 - \gamma((10 - x) + (10 - y))$ is the true utility, and is linear.

b. Same as in a, except that $U(10, 1) = -1$.

c. The exact utility depends on the exact placement of the obstacles. The best approximation is the same as in a. The features in exercise 21.9 might improve the approximation.

d. The optimal policy is to head straight for the goal from any point on the right side of the wall, and to head for (5, 10) first (and then for the goal) from any point on the left of the wall. Thus, the exact utility function is:

$$\begin{aligned} U(x, y) &= 1 - \gamma((10 - x) + (10 - y)) \qquad (\text{if } x \geq 5) \\ &= 1 - \gamma((5 - x) + (10 - y)) - 5\gamma \qquad (\text{if } x < 5) \end{aligned}$$

Unfortunately, this is not linear in x and y, as stated. Fortunately, we can restate the optimal policy as "head straight up to row 10 first, then head right until column 10." This gives us the same exact utility as in a, and the same linear approximation.

e. $U(x, y) = 1 - \gamma(|5 - x| + |5 - y|)$ is the true utility. This is also not linear in x and y, because of the absolute value signs. All can be fixed by introducing the features $|5 - x|$ and $|5 - y|$.

21.9 Code not shown.

21.10 To map evolutionary processes onto the formal model of reinforcement learning, one must find evolutionary analogs for the reward signal, learning process, and the learned policy. Let us start with a simple animal that does not learn during its own lifetime. This animal's genotype, to the extent that it determines animal's behavior over its lifetime, can be thought of as the parameters θ of a policy pi_θ. Mutations, crossover, and related processes are the

*距离最近-1终端状态。 *相邻+1终端状态的数量。 *相邻-1终端状态的数量。 *相邻障碍物的数量。 *与最近+1终端状态的路径相交的障碍物数。

21.7修改涉及结合游戏环境转换器的元素（lisp/search/games中的游戏->环境。lisp）与功能mdp->环境的元素。 奖励信号只是获胜/绘图/失败的效用，仅在游戏结束时发生。 每个代理使用的评估函数是它通过TD过程学习的效用函数。 保持TD学习过程（完全独立于正在玩游戏的事实）与游戏算法不同是很重要的。 将评估函数与深度搜索一起使用可能会更好，因为它将通过提高游戏质量来帮助代理专注于搜索空间的相关部分。 然而，有一个权衡：搜索越深，在玩每个训练游戏中使用的计算机时间就越多。

21.8 这是一个相对耗时的练习。 未显示的代码计算三维图。 实用程序功能是:

a. U(x,y)=1-γ((10-x)+(10-y))是真正的效用，并且是线性的。 b. 与a中相同，不同之处在于U(10,1)=-1。 c. 确切的效用取决于障碍物的确切位置。 最佳近似值与a中相同。练习21.9中的特征可能会改进近似值。 d. 最佳策略是从墙右侧的任何点直奔目标，并从墙左侧的任何点首先前往（5，10）（然后前往目标）。 因此，确切的效用函数是:

$$\begin{aligned} U(x, y) &= 1 - \gamma((10 - x) + (10 - y)) \qquad (\text{if } x \geq 5) \\ &= 1 - \gamma((5 - x) + (10 - y)) - 5\gamma \qquad (\text{if } x < 5) \end{aligned}$$

不幸的是，这在x和y中不是线性的，如所述。 幸运的是，我们可以将最优策略重申为"先直走到第10行，然后向右走到第10列。"这给了我们与a中相同的确切效用，以及相同的线性近似。 e. U(x,y)=1-γ(/5-x| + | 5 -y/)是真正的效用。 这在x和y中也不是线性的，因为绝对值符号。 所有这些都可以通过引入功能|5-x|和|5-y/来修复。

21.9 未示出的代码。

21.10要将进化过程映射到强化学习的正式模型上，必须找到奖励信号，学习过程和学习政策的进化类似物。 让我们从一个在自己的一生中不学习的简单动物开始。 这种动物的基因型，在一定程度上决定了动物在其一生中的行为，可以被认为是政策pi θ的参数θ。 突变、交叉和相关过程是

part of the learning algorithm—like an empirical gradient neighborhood generator in policy search—that creates new values of θ. One can also imagine a reinforcement learning process that works on many different copies of π simultaneously, as evolution does; evolution adds the complication that each copy of π modifies the environment for other copies of π, whereas in RL the environment dynamics are assumed fixed, independent of the policy chosen by the agent. The most difficult issue, as the question indicates, is the reward function and the underlying objective function of the learning process. In RL, the objective function is to find policies that maximize the expected sum of rewards over time. Biologists usually talk about evolution as maximizing "reproductive fitness," i.e., the ability of individuals of a given genotype to reproduce and thereby propagate the genotype to the next generation. In this simple view, evolution's "objective function" is to find the π that generates the most copies of itself over infinite time. Thus, the "reward signal" is positive for creation of new individuals; death, *per se*, seems to be irrelevant.

Of course, the real story is much more complex. Natural selection operates not just at the genotype level but also at the level of individual genes and groups of genes; the environment is certainly multiagent rather than single-agent; and, as noted in the case of Baldwinian evolution in Chapter 4, evolution may result in organisms that have hardwired reward signals that are related to the fitness reward and may use those signals to learn during their lifetimes.

As far as we know there has been no careful and philosophically valid attempt to map evolution onto the formal model of reinforcement learning; any such attempt must be careful not to *assume* that such a mapping is possible or to *ascribe* a goal to evolution; at best, one may be able to interpret what evolution tends to do *as if* it were the result of some maximizing process, and ask what it is that is being maximized.

199

学习算法的一部分—就像策略搜索中的经验梯度邻域生成器-创建θ的新值。 人们也可以想象一个强化学习过程，它同时在π的许多不同副本上工作，就像进化一样；进化增加了一个复杂的问题，即π的每个副本都修改了π的其他副本的环境，而在RL中，环境动力学被假定为固定的，独立于代理所选择的策略。 正如问题所示，最困难的问题是奖励函数和学习过程的基本目标函数。 在RL中，目标函数是找到随时间最大化预期奖励总和的策略。 生物学家通常将进化称为最大化"生殖健康"，即给定基因型的个体繁殖并由此将基因型传播到下一代的能力。 在这个简单的观点中，进化的"目标函数"是找到在无限时间内产生最多副本的π。 因此，"奖励信号"对于创造新的个体是积极的;死亡本身似乎是无关紧要的。

当然，真实的故事要复杂得多。 自然选择不仅在基因型水平上运作，而且在单个基因和基因组的水平上运作；环境肯定是多因子的，而不是单因子的；而且，正如第四章巴尔德维尼亚进化的例子所述，进化可能导致有机体具有与健康奖励有关的硬连线奖励信号，并可能利用这些信号在有生之年学习。 据我们所知，还没有仔细和哲学上有效的尝试将进化映射到强化学习的正式模型上；任何这样的尝试都必须小心，不要假设这样的映射是可能的，或者将目标归因于进化；充其量，人们可能能够解释进化倾向于做什么，好像它是某种最大化过程的结果，并询问它是什么被最大化。

Solutions for Chapter 22
Natural Language Processing

22.1 Code not shown. The distribution of words should fall along a Zipfian distribution: a straight line on a log-log scale. The generated language should be similar to the examples in the chapter.

22.2 Using a unigram language model, the probability of a segmentation of a string $s_{1:N}$ into k nonempty words $s = w_1 \dots w_k$ is $\prod_{i=1}^{k} P_{lm}(w_i)$ where P_{lm} is the unigram language model. This is not normalized without a distribution over the number of words k, but let's ignore this for now.

To see that we can find the most probable segmentation of a string by dynamic programming, let $p(i)$ be the maximum probability of any segmentation of $s_{i:N}$ into words. Then $p(N+1) = 1$ and

$$p(i) = \max_{j=i,\dots,N} P_{lm}(s_{i:j})\, p(j+1)$$

because any segmentation of $s_{i:N}$ starts with a single word spanning $s_{i:j}$ and a segmentation of the rest of the string $s_{j+1:N}$. Because we are using a unigram model, the optimal segmentation of $s_{j+1:N}$ does not depend on the earlier parts of the string.

Using the techniques of this chapter to form a unigram model accessed by the function `prob_word(word)`, the following Python code solves the above dynamic program to output an optimal segmentation:

```
def segment(text):
    length = len(text)
    max_prob = [0] * (length+1)
    max_prob[length] = 1
    split_idx = [-1] * (length+1)
    for start in range(length,-1,-1):
        for split in range(start+1,length+1):
            p = max_prob[split] * prob_word(text[start:split])
            if p > max_prob[start]:
                max_prob[start] = p
                split_idx[start] = split
    i = 0
    words = []
    while i < length:
        words.append(text[i:split_idx[i]])
        i = split_idx[i]
        if i == -1:
```


第22章的解决方案
自然语言处理

22.1未显示的代码。 单词的分布应该沿着Zipfian分布：对数-对数尺度上的直线。 生成的语言应该类似于章节中的示例。

22.2使用unigram语言模型，将一个字符串s1:N分割成k个非空词的概率s=w1。 . . w i=1P lm(w i)其中P lm是unigram语言模型。 如果没有单词数k的分布，这不是规范化的，但现在让我们忽略这一点。

为了看到我们可以通过动态pro语法找到最可能的字符串分割，让p（i）是s i：N任何分割成单词的最大概率。 则p(N+1)=1和

$$p(i) = \text{max j=i，。.„N P lm(s i:j)p(j+1)}$$

因为s i:N的任何分段都以跨越s i:j的单个单词和字符串s j+1:N的其余部分的分段开始。 因为我们使用的是unigram模型，所以s j+1：N的最佳分割不依赖于字符串的早期部分。

利用本章的技术形成函数prob_word(word)访问的unigram模型，下面的Python代码对上述动态程序进行求解，输出一个最优的分割:

```
def segment(text):
     length=len(text)max_prob=[0]*(length+1)m
    ax_prob[length]=1split_idx=[-1]*(length+1)f
    or start in range(length,-1,-1):


        对于范围分割（开始+1，长度+1）：
            p=max_prob[split]*prob_word(text[start:split])如果p>max_prob[sta
            rt]:max_prob[start]=p split_idx[start]=split


    i=0字=[]而i<长度:


        words.append(text[i:split_idx[i]])
        i = split_idx[i]
        if i == -1:
```

```
            return None  # for text with zero probability
    return words
```

One caveat is the language model must assign probabilities to unknown words based on their length, otherwise sufficiently long strings will be segmented as single unknown words. One natural option is to fit an exponential distribution to the words lengths of a corpus. Alternatively, one could learn a distribution over the number of words in a string based on its length, add a $P(k)$ term to the probability of a segmentation, and modify the dynamic program to handle this (i.e., to compute $p(i, k)$ the maximum probability of segmenting $s_{i:N}$ into k words).

22.3 Code not shown. The approach suggested here will work in some cases, for authors with distinct vocabularies. For more similar authors, other features such as bigrams, average word and sentence length, parts of speech, and punctuation might help. Accuracy will also depend on how many authors are being distinguished. One interesting way to make the task easier is to group authors into male and female, and try to distinguish the sex of an author not previously seen. This was suggested by the work of Shlomo Argamon.

22.4 Code not shown. There are now several open-source projects to do Bayesian spam filtering, so beware if you assign this exercise.

22.5 Doing the evaluation is easy, if a bit tedious (requiring 150 page evaluations for the complete 10 documents $\times$ 3 engines $\times$ 5 queries). Explaining the differences is more difficult. Some things to check are whether the good results in one engine are even in the other engines at all (by searching for unique phrases on the page); check whether the results are commercially sponsored, are produced by human editors, or are algorithmically determined by a search ranking algorithm; check whether each engine does the features mentioned in the next exercise.

22.6 One good way to do this is to first find a search that yields a single page (or a few pages) by searching for rare words or phrases on the page. Then make the search more difficult by adding a variant of one of the words on the page—a word with different case, different suffix, different spelling, or a synonym for one of the words on the page, and see if the page is still returned. (Make sure that the search engine requires all terms to match for this technique to work.)

22.7 Code not shown. The simplest approach is to look for a string of capitalized words, followed by “Inc” or “Co.” or “Ltd.” or similar markers. A more complex approach is to get a list of company names (e.g. from an online stock service), look for those names as exact matches, and also extract patterns from them. Reporting recall and precision requires a clearly-defined corpus.

22.8

A. Use the precision on the first 20 documents returned.
B. Use the reciprocal rank of the first relevant document. Or just the rank, considered as a cost function (large is bad).
C. Use the recall.

201

```
            不返回            #对于概率为零的文本
    返回单词
```

一个警告是，语言模型必须根据未知单词的长度为其分配概率，否则足够长的字符串将被分割为单个未知单词。 一个自然的选择是将指数分布拟合到语料库的单词长度。 或者，可以根据字符串的长度学习字符串中单词数量的分布，在分割概率上添加P（k）项，并修改动态程序来处理这个问题（即计算p（i，k）将s i：N分割成k个单词的最大概率）。

22.3未显示的代码。 这里建议的方法在某些情况下适用于具有不同词汇的作者。 对于更相似的作者，其他功能，如bigrams，平均单词和句子长度，词性和标点符号可能会有所帮助。 准确性还将取决于有多少作者被区分。 使任务更容易的一个有趣的方法是将作者分为男性和女性，并尝试区分以前未见过的作者的性别。 这是由Shlomo Argamon的工作提出的。

22.4未显示的代码。 现在有几个开源项目可以做贝叶斯垃圾邮件过滤，所以如果你分配这个练习，请小心。

22.5做评估很容易，如果有点乏味（完整的10个文档×3个引擎×5个查询需要150页的评估）。 解释差异更多的是diffi邪教。 要检查的一些事情是，一个引擎中的好结果是否甚至在其他引擎中（通过在页面上搜索唯一的短语）;检查结果是否是商业赞助的，是由人类编辑制作的，还是由搜索排名算法在算法上确定的;检查每个引擎是否执行下一个练习中提到的功能。

22.6这样做的一个好方法是首先通过在页面上搜索生僻的单词或短语来找到一个产生单个页面（或几页）的搜索。 然后通过添加页面上其中一个单词的变体—具有不同大小写，不同后缀，不同拼写或页面上其中一个单词的同义词的单词，使搜索变得更加困难，并查看页 （确保搜索引擎要求所有术语匹配此技术的工作。）

22.7未显示的代码。 最简单的方法是寻找一串大写的单词，然后是"Inc"或"Co"。"或"有限公司"或类似标记。 更复杂的方法是获取公司名称列表（例如从在线股票服务），查找这些名称作为完全匹配，并从中提取模式。 报告召回和精确性需要一个明确定义的语料库。

22.8

A.使用返回的前20个文档的精度。
B.使用第一个相关文档的倒数等级。 或者只是排名，被视为成本函数（大是坏的）。 C.使用召回。

D. Score this as 1 if the first 100 documents retrieved contain at least one relevant to the query and 0 otherwise.

E. Score this as $(A(R + I) + BR - NC) where R is the number of relevant documents retrieved, I is the number of irrelevant documents retrieved, and C is the number of relevant documents not retrieved.

F. One model would be a probabilistic one, in which, if the user has seen R relevant documents and I irrelevant ones, she will continue searching with probability $p(R, I)$ for some function p, to be specified. The measure of quality is then the expected number of relevant documents examined.

如果检索到的前100个文档至少包含一个与查询相关的文档，则将其评分为1，否则为0。 E.将其评分为$(A(R+I)+BR-NC)其中R是检索到的相关文档的数量，I是检索到的不相关文档的数量，C是未检索到的相关文档的数量。 F.一个模型将是一个概率模型，其中，如果用户已经看到r个相关文档和i个不相关的文档，她将继续以概率p（R，I）搜索某些函数p，以指定。 然后，质量的衡量标准是检查的相关文件的预期数量。

Solutions for Chapter 23
Natural Language for Communication

23.1 No answer required; just read the passage.

23.2 The prior is represented by rules such as

$$P(N_0 = A): \qquad S \rightarrow A\ S_A$$

where S_A means "rest of sentence after an A." Transitions are represented as, for example,

$$P(N_{t+1} = B \mid N_t = A): \qquad S_A \rightarrow B\ S_B$$

and the sensor model is just the lexical rules such as

$$P(W_t = \text{is} \mid N_t = A): \qquad A \rightarrow \text{is}\ .$$

23.3

a. (i).

b. This has two parses. The first uses $VP \rightarrow VP\ Adverb$, $VP \rightarrow Copula\ Adjective$, $Copula \rightarrow is$, $Adjective \rightarrow$ **well**, $Adverb \rightarrow$ **well**. Its probability is

$$0.2 \times 0.2 \times 0.8 \times 0.5 \times 0.5 = 0.008\ .$$

The second uses $VP \rightarrow VP\ Adverb$ twice, $VP \rightarrow Verb$, $Verb \rightarrow is$, and $Adverb \rightarrow$ **well** twice. Its probability is

$$0.2 \times 0.2 \times 0.1 \times 0.5 \times 0.5 \times 0.5 = 0.0005\ .$$

The total probability is 0.0085.

c. It exhibits both lexical and syntactic ambiguity.

d. True. There can only be finitely many ways to generate the finitely many strings of 10 words.

23.4 The purpose of this exercise is to get the student thinking about the properties of natural language. There is a wide variety of acceptable answers. Here are ours:

- **Grammar and Syntax** Java: formally defined in a reference book. Grammaticality is crucial; ungrammatical programs are not accepted. English: unknown, never formally defined, constantly changing. Most communication is made with "ungrammatical" utterances. There is a notion of graded acceptability: some utterances are judged slightly ungrammatical or a little odd, while others are clearly right or wrong.

第23章的解决方案
交流的自然语言

23.1 不需要回答，只要读一读这段话就行了。

23.2 先验由诸如

$$P(N_0 = A): \qquad S \rightarrow A\ S_A$$

其中S a表示"a之后的句子的其余部分。"过渡表示为，例如,

$$P(N_{t+1} = B \mid N_t = A): \qquad S_A \rightarrow B\ S_B$$

而传感器模型只是词汇规则，如

$$P(W_t = \text{is} \mid N_t = A): \qquad A \rightarrow \text{is}\ .$$

23.3

a. (i)。 b. 这有两个解析。 第一个使用V P→V P副词，V P→Copula形容词，Copula→is，形容词→well，副词→well。 它的概率是

$$0.2 \times 0.2 \times 0.8 \times 0.5 \times 0.5 = 0.008\ .$$

第二个使用V P→V P副词两次，V P→V erb，V erb→is，副词→well两次。 它的概率是

$$0.2 \times 0.2 \times 0.1 \times 0.5 \times 0.5 \times 0.5 = 0.0005\ .$$

总概率为0.0085。 c. 它同时表现出词汇和句法上的歧义. d. 没错。 只能有很多方法来生成10个单词的很多字符串。

23.4本练习的目的是让学生思考自然语言的属性。 有各种各样的可接受的答案。 这是我们的:

*语法和语法Java：在参考书中正式定义。 语法性是至关重要的;不接受非语法程序。 中文:未知,从未正式定义,不断变化. 大多数沟通都是用"不语法"的话语进行的。 有一个分级可接受性的概念：一些话语被判断为稍微不合语法或有点奇怪，而另一些话语显然是对的或错的。

- **Semantics** Java: the semantics of a program is formally defined by the language specification. More pragmatically, one can say that the meaning of a particular program is the JVM code emitted by the compiler. English: no formal semantics, meaning is context dependent.
- **Pragmatics and Context-Dependence** Java: some small parts of a program are left undefined in the language specification, and are dependent on the computer on which the program is run. English: almost everything about an utterance is dependent on the situation of use.
- **Compositionality** Java: almost all compositional. The meaning of "A + B" is clearly derived from the meaning of "A" and the meaning of "B" in isolation. English: some compositional parts, but many non-compositional dependencies.
- **Lexical Ambiguity** Java: a symbol such as "Avg" can be locally ambiguous as it might refer to a variable, a class, or a function. The ambiguity can be resolved simply by checking the declaration; declarations therefore fulfill in a very exact way the role played by background knowledge and grammatical context in English. English: much lexical ambiguity.
- **Syntactic Ambiguity** Java: the syntax of the language resolves ambiguity. For example, in "if (X) if (Y) A; else B;" one might think it is ambiguous whether the "else" belongs to the first or second "if," but the language is specified so that it always belongs to the second. English: much syntactic ambiguity.
- **Reference** Java: there is a pronoun "this" to refer to the object on which a method was invoked. Other than that, there are no pronouns or other means of indexical reference; no "it," no "that." (Compare this to stack-based languages such as Forth, where the stack pointer operates as a sort of implicit "it.") There is reference by name, however. Note that ambiguities are determined by scope—if there are two or more declarations of the variable "X", then a use of X refers to the one in the innermost scope surrounding the use. English: many techniques for reference.
- **Background Knowledge** Java: none needed to interpret a program, although a local "context" is built up as declarations are processed. English: much needed to do disambiguation.
- **Understanding** Java: understanding a program means translating it to JVM byte code. English: understanding an utterance means (among other things) responding to it appropriately; participating in a dialog (or choosing not to participate, but having the potential ability to do so).

As a follow-up question, you might want to compare different languages, for example: English, Java, Morse code, the SQL database query language, the Postscript document description language, mathematics, etc.

23.5 The purpose of this exercise is to get some experience with simple grammars, and to see how context-sensitive grammars are more complicated than context-free. One approach to writing grammars is to write down the strings of the language in an orderly fashion, and then see how a progression from one string to the next could be created by recursive application of rules. For example:

*语义Java：程序的语义由语言规范正式定义。 更务实地说，可以说特定程序的含义是编译器发出的JVM代码。 中文:没有正式的语义,意义是上下文相关的.*语用学和上下文依赖Java：程序的一些小部分在语言规范中没有定义，并且依赖于运行程序的计算机。 中文:几乎所有关于话语的事情都取决于使用的情况.*组合性Java：几乎所有的组合性。 "A+B"的含义显然是孤立地从"A"的含义和"B"的含义中派生出来的。 英文：some compositional parts，but many non-compositional dependencies。 *词汇歧义Java：像"Avg"这样的符号可能是局部歧义的，因为它可能指的是变量、类或函数。 模糊性可以通过检查声明来解决;因此，声明以非常精确的方式履行背景知识和语法上下文在英语中所起的作用。 中文:词汇多义性.*句法歧义Java：语言的语法解决歧义。 例如，在"if(X)if(Y)A;else B;"中，人们可能会认为"else"属于第一个还是第二个"if"是模棱两可的，但语言被指定为始终属于第二个。 中文:语法模糊.*引用Java：有一个代词"this"来引用调用方法的对象。 除此之外，没有代词或其他索引引用的手段;没有"它"，没有"那个。"（将其与基于堆栈的语言进行比较，例如Forth，其中堆栈指针作为一种隐式"it。"）有按名称引用，但是。 请注意，歧义是由范围决定的-如果变量"X"有两个或多个声明，那么X的使用是指围绕使用的最内部范围中的一个。 中文：许多技术供参考。 *背景知识Java：不需要解释一个程序，尽管在处理声明时建立了一个本地"上下文"。 L1:非常需要做消歧。 *理解Java：理解一个程序意味着将其翻译为JVM字节代码。 英语：理解一个话语意味着（除其他事项外）适当地回应它;参与对话（或选择不参与，但具有这样做的潜在能力）。 作为后续问题，您可能需要比较不同的语言，例如：英语，Java，莫尔斯电码，SQL数据库查询语言，Postscript文档描述语言，数学等。

23.5本练习的目的是获得一些简单语法的经验，并了解上下文相关语法如何比上下文无关更复杂。 编写语法的一种方法是有序地写下语言的字符串，然后查看如何通过递归应用规则来创建从一个字符串到下一个字符串的过程。 例如:

a. The language $a^n b^n$: The strings are ϵ, ab, $aabb$, ...(where ϵ indicates the null string). Each member of this sequence can be derived from the previous by wrapping an a at the start and a b at the end. Therefore a grammar is:

$$S \rightarrow \epsilon$$
$$S \rightarrow a\ S\ b$$

b. The palindrome language: Let's assume the alphabet is just a, b and c. (In general, the size of the grammar will be proportional to the size of the alphabet. There is no way to write a context-free grammar without specifying the alphabet/lexicon.) The strings of the language include ϵ, *a, b, c, aa, bb, cc, aaa, aba, aca, bab, bbb, bcb, ...*. In general, a string can be formed by bracketing any previous string with two copies of any member of the alphabet. So a grammar is:

$$S \rightarrow \epsilon \mid a \mid b \mid c \mid a\ S\ a \mid b\ S\ b \mid c\ S\ c$$

c. The duplicate language: For the moment, assume that the alphabet is just ab. (It is straightforward to extend to a larger alphabet.) The duplicate language consists of the strings: ϵ, aa, bb, $aaaa$, $abab$, $bbbb$, $baba$, ...Note that all strings are of even length.

One strategy for creating strings in this language is this:

- Start with markers for the front and middle of the string: we can use the non-terminal F for the front and M for the middle. So at this point we have the string FM.
- Generate items at the front of the string: generate an a followed by an A, or a b followed by a B. Eventually we get, say, $FaAaAbBM$. Then we no longer need the F marker and can delete it, leaving $aAaAbBM$.
- Move the non-terminals A and B down the line until just before the M. We end up with $aabAABM$.
- Hop the As and Bs over the M, converting each to a terminal (a or b) as we go. Then we delete the M, and are left with the end result: $aabaab$.

Here is a grammar to implement this strategy:

$S \rightarrow F\ M$	(starting markers)
$F \rightarrow F\ a\ A$	(introduce symbols)
$F \rightarrow F\ b\ B$	
$F \rightarrow \epsilon$	(delete the F marker)
$A\ a \rightarrow a\ A$	(move non-terminals down to the M)
$A\ b \rightarrow b\ A$	
$B\ a \rightarrow a\ B$	
$B\ b \rightarrow b\ B$	
$A\ M \rightarrow M\ a$	(hop over M and convert to terminal)
$B\ M \rightarrow M\ b$	
$M \rightarrow \epsilon$	(delete the M marker)

Here is a trace of the grammar deriving $aabaab$:

$\underline{S}$

a. 语言a n b n:字符串是 ,ab,aabb,...（其中 表示空字符串）。 这个序列的每个成员都可以通过在开始处包裹一个a，在结束处包裹一个b来从前面派生出来。 因此，语法是:

$$S \rightarrow \epsilon$$
$$S \quad a\ S\ b$$

b. 回文语言：假设字母表只是a，b和c。 （一般来说，语法的大小将与字母表的大小成正比。 如果不指定字母/词汇，就无法编写上下文无关的语法。）语言的字符串包括 ，a，b，c，aa，bb，cc，aaa，aba，aca，bab，bbb，bcb，。...一般来说，字符串可以通过用字母表中任何成员的两个副本包围任何先前的字符串来形成。 所以语法是:

$$S \rightarrow \epsilon \mid a \mid b \mid c \mid a\ S\ a \mid b\ S\ b \mid c\ S\ c$$

c. 重复的语言：目前，假设字母表只是ab。 （扩展到更大的字母表很简单。）重复的语言由字符串组成： ，aa，bb，aaaa，abab，bbbb，baba，。..请注意，所有字符串都是偶数长度。 用这种语言创建字符串的一种策略是:

•从字符串前面和中间的标记开始：我们可以使用非终结符F表示前面，M表示中间。 所以在这一点上，我们有字符串FM。 *在字符串的前面生成项：生成一个a后跟一个A，或一个b后跟一个B。 最终我们得到，比如，FaAaAbBM。 然后我们不再需要F标记，可以删除它，留下aAaAbBM。 *将非端子A和B向下移动直到M之前。 我们最终得到了aabAABM。 *将a s和B s跳过M，将每个转换为终端（a或b）。 然后我们删除M，并留下最终结果：aabaab。

以下是实现此策略的语法:

$S \rightarrow F\ M$	(starting markers)
$F \rightarrow F\ a\ A$	(introduce symbols)
$F \rightarrow F\ b\ B$	
$F \rightarrow \epsilon$	(delete the F marker)
$A\ a \rightarrow a\ A$	（将非终端向下移动到M）
$A\ b \rightarrow b\ A$	
$B\ a \rightarrow a\ B$	
$B\ b \quad b\ B$	
A M→M A(跳过M并转换为终端)	
$M \quad \epsilon$	(delete the M marker)

以下是派生aabaab的语法的痕迹 :

$\underline{S}$

$\underline{F}M$
$\underline{F}bBM$
$\underline{F}aAbBM$
$\underline{F}aAaAbBM$
$a\underline{Aa}AbBM$
$aaA\underline{Ab}BM$
$aa\underline{Ab}ABM$
$aabAA\underline{BM}$
$aabA\underline{AM}b$
$aab\underline{AM}ab$
$aab\underline{M}aab$
$aabaab$

23.6 Grammar (A) does not work, because there is no way for the verb "walked" followed by the adverb "slowly" and the prepositional phrase "to the supermarket" to be parsed as a verb phrase. A verb phrase in (A) must have either two adverbs or be just a verb. Here is the parse under grammar (B):

```
S---NP-+-Pro---Someone
       |
       |-VP-+-V---walked
            |
            |-Vmod-+-Adv---slowly
                   |
                   |-Vmod---Adv---PP---Prep-+-to
                                            |
                                            |-NP-+-Det---the
                                                 |
                                                 |-NP---Noun---supermarket
```

Here is the parse under grammar (C):

```
S---NP-+-Pro---Someone
       |
       |-VP-+-V---walked
            |
            |-Adv-+-Adv---slowly
                  |
                  |-Adv---PP---Prep-+-to
                                    |
                                    |-NP-+-Det---the
                                         |
                                         |-NP---Noun---supermarket
```

23.7 Here is a start of a grammar:

```
Time => DigitHour ":" DigitMinute
    | "midnight" | "noon" | "12 midnight" | "12 noon''
    | ClockHour "o'clock"
```


$\underline{F}M$
$\underline{F}bBM$
$\underline{F}aAbBM$
$\underline{F}aAaAbBM$
$a\underline{Aa}AbBM$
$aaA\underline{Ab}BM$
$aa\underline{Ab}ABM$
$aabAA\underline{BM}$
$aabA\underline{AM}b$
$aab\underline{AM}ab$
$aab\underline{M}aab$
$aabaab$

23.6语法（A）不起作用，因为动词"walked"后跟副词"slowly"和介词短语"to the supermarket"无法被解析为动词短语。（A）中的动词短语必须有两个副词，或者只是一个动词。下面是语法（B）下的解析:

```
S---NP-+-Pro---Someone
       |
       |-VP-+-V---walked
            |
            |-Vmod-+-Adv---slowly
                   |
                   |-Vmod---Adv---PP---Prep-+-to
                                            |
                                            |-NP-+-Det---the
                                                 |
                                                 |-NP---Noun---supermarket
```

下面是语法（C）下的解析:

```
S---NP-+-Pro---Someone
       |
       |-VP-+-V---walked
            |
            |-Adv-+-Adv---slowly
                  |
                  |-Adv---PP---Prep-+-to
                                    |
                                    |-NP-+-Det---the
                                         |
                                         |-NP---Noun---supermarket
```

23.7 这里是一个语法的开始:

```
Time => DigitHour ":" DigitMinute
    | "midnight" | "noon" | "12 midnight" | "12 noon''
    | ClockHour "o'clock"
```

$$
\begin{array}{rcl}
S & \rightarrow & NP\ VP \\
 & | & S'\ Conj\ S' \\
S' & \rightarrow & NP\ VP \\
 & | & SConj\ S' \\
SConj & \rightarrow & S'\ Conj \\
 & & \\
NP & \rightarrow & \textbf{me} \mid \textbf{you} \mid \textbf{I} \mid \textbf{it} \mid \ldots \\
 & | & \textbf{John} \mid \textbf{Mary} \mid \textbf{Boston} \mid \ldots \\
 & | & \textbf{stench} \mid \textbf{breeze} \mid \textbf{wumpus} \mid \textbf{pits} \mid \ldots \\
 & | & Article\ Noun \\
 & | & ArticleAdjs\ Noun \\
 & | & Digit\ Digit \\
 & | & NP\ PP \\
 & | & NP\ RelClause \\
ArticleAdjs & \rightarrow & Article\ Adjs \\
 & & \\
VP & \rightarrow & \textbf{is} \mid \textbf{feel} \mid \textbf{smells} \mid \textbf{stinks} \mid \ldots \\
 & | & VP\ NP \\
 & | & VP\ Adjective \\
 & | & VP\ PP \\
 & | & VP\ Adverb \\
 & & \\
Adjs & \rightarrow & \textbf{right} \mid \textbf{dead} \mid \textbf{smelly} \mid \textbf{breezy} \ldots \\
 & | & Adjective\ Adjs \\
PP & \rightarrow & Prep\ NP \\
RelClause & \rightarrow & RelPro\ VP
\end{array}
$$

Figure S23.1 The final result after turning $\mathcal{E}_0$ into CNF (omitting probabilities).

```
    | Difference BeforeAfter ExtendedHour

DigitHour => 0 | 1 | ... | 23
DigitMinute => 1 | 2 | ... | 60
HalfDigitMinute => 1 | 2 | ... | 29
ClockHour => ClockDigitHour | ClockWordHour
ClockDigitHour => 1 | 2 | ... | 12
ClockWordHour => "one" | ... | "twelve"
BeforeAfter =>  "to" | "past" | "before" | "after"
Difference => HalfDigitMinute "minutes" | ShortDifference
ShortDifference => "five" | "ten" | "twenty" | "twenty-five" | "quarter" | "half"
ExtendedHour => ClockHour | "midnight" | "noon"
```

The grammar is not perfect; for example, it allows “ten before six” and “quarter past noon,” which are a little odd-sounding, and “half before six,” which is not really OK.

207

$$
\begin{array}{rcl}
S & \rightarrow & NP\ VP \\
 & | & S'\ Conj\ S' \\
S' & \rightarrow & NP\ VP \\
 & | & SConj\ S' \\
SConj & \rightarrow & S'\ Conj \\
 & & \\
NP & \rightarrow & \text{我/你/我/它/。..} \\
 & | & \text{约翰/玛丽/波士顿/...} \\
 & | & \text{恶臭/微风|wumpus/pits|。..} \\
 & | & \text{文章名词} \\
 & | & ArticleAdjs\ Noun \\
 & | & Digit\ Digit \\
 & | & NP\ PP \\
 & | & NP\ RelClause \\
ArticleAdjs & \rightarrow & Article\ Adjs \\
 & & \\
VP & \rightarrow & \text{是/感觉/气味|臭/。..} \\
 & | & \\
 & | & VP\ Adjective \\
 & | & VP\ PP \\
 & | & VP\ Adverb \\
 & & \\
Adjs & \rightarrow & \text{右/死/臭/微风。..} \\
 & | & \\
PP & \rightarrow & Prep\ NP \\
RelClause & \rightarrow & RelPro\ VP
\end{array}
$$

Figure S23.1 将E0变为CNF（省略概率）后的最终结果。


```
    | Difference BeforeAfter ExtendedHour

DigitHour => 0 | 1 | ... | 23
DigitMinute => 1 | 2 | ... | 60
HalfDigitMinute => 1 | 2 | ... | 29
ClockHour => ClockDigitHour | ClockWordHour
ClockDigitHour => 1 | 2 | ... | 12
ClockWordHour => "one" | ... | "twelve"
BeforeAfter =>  "to" | "past" | "before" | "after"
Difference => HalfDigitMinute "minutes" | ShortDifference
ShortDifference => "five" | "ten" | "twenty" | "twenty-five" | "quarter" | "half"
ExtendedHour => ClockHour | "midnight" | "noon"
```


语法并不完美;例如，它允许"六点前十点"和"中午一刻过去"，这有点听起来很奇怪，还有"六点前一半"，这并不是很好。

23.8 The final grammar is shown in Figure S23.1. (Note that in early printings, the question asked for the rule $S' \rightarrow S$ to be added.) In step **d**, students may be tempted to drop the rules ($Y \rightarrow \ldots$), which fails immediately.

$S \rightarrow NP(Subjective, number, person)\ VP(number, person) \mid \ldots$
$NP(case, number, person) \rightarrow Pronoun(case, number, person)$
$NP(case, number, Third) \rightarrow Name(number) \mid Noun(number) \mid \ldots$
$VP(number, person) \rightarrow VP(number, person)\ NP(Objective, \_, \_) \mid \ldots$
$PP \rightarrow Preposition\ NP(Objective, \_, \_)$
$Pronoun(Subjective, Singular, First) \rightarrow$ **I**
$Pronoun(Subjective, Singular, Second) \rightarrow$ **you**
$Pronoun(Subjective, Singular, Third) \rightarrow$ **he** | **she** | **it**
$Pronoun(Subjective, Plural, First) \rightarrow$ **we**
$Pronoun(Subjective, Plural, Second) \rightarrow$ **you**
$Pronoun(Subjective, Plural, Third) \rightarrow$ **they**
$Pronoun(Objective, Singular, First) \rightarrow$ **me**
$Pronoun(Objective, Singular, Second) \rightarrow$ **you**
$Pronoun(Objective, Singular, Third) \rightarrow$ **him** | **her** | **it**
$Pronoun(Objective, Plural, First) \rightarrow$ **us**
$Pronoun(Objective, Plural, Second) \rightarrow$ **you**
$Pronoun(Objective, Plural, Third) \rightarrow$ **them**
$Verb(Singular, First) \rightarrow$ **smell**
$Verb(Singular, Second) \rightarrow$ **smell**
$Verb(Singular, Third) \rightarrow$ **smells**
$Verb(Plural, \_) \rightarrow$ **smell**

Figure S23.2 A partial DCG for $\mathcal{E}_1$, modified to handle subject–verb number/person agreement as in Ex. 22.2.

23.9 See Figure S23.2 for a partial DCG. We include both person and number annotation although English really only differentiates the third person singular for verb agreement (except for the verb *be*).

23.10 One parse captures the meaning "I am able to fish" and the other "I put fish in cans." Both have the left branch $NP \rightarrow Pronoun \rightarrow$ **I**, which has probability 0.16.

- The first has the right branch $VP \rightarrow Modal\,Verb$ (0.2) with $Modal \rightarrow$ **can** (0.3) and $Verb \rightarrow$ **fish** (0.1), so its prior probability is

$$0.16 \times 0.2 \times 0.3 \times 0.1 = 0.00096\,.$$

- The second has the right branch $VP \rightarrow Verb\,NP$ (0.8) with $Verb \rightarrow$ **can** (0.1) and $NP \rightarrow Noun \rightarrow$ **fish** (0.6×0.3), so its prior probability is

$$0.16 \times 0.8 \times 0.1 \times 0.6 \times 0.3 = 0.002304\,.$$

23.8最终语法如图S23.1所示。（请注意，在早期打印中，问题要求添加规则S'→S。）在步骤d中，学生可能会试图放弃规则（Y→。..），其立即失效。

S→NP(人、数、人)VP(数、人)|。..
NP(case,number,person)→代词(case,number,person)
NP(case,number,Third)→Name(数字)|Noun(数字)|。..
VP(number,person)→VP(number,person)NP(Objective,,)/。..
PP→介词NP(Objective,,)
代词（主观，单数，第一）→I
代词（主观，单数，第二）→你
代词（主观，单数，第三）→他|她/它
代词（主观，复数，第一）→我们
代词（主观，复数，第二）→你代词（主观
，复数，第三）→他们代词（客观，单数，
第一）→我
代词（客观，单数，第二）→你
代词（客观，单数，第三）→他|她/它
代词（目标，复数，第一）→我们
代词（客观，复数，第二）→你代词（客
观，复数，第三）→他们
动词（单数，第一）→气味
动词（单数，第二）→气味动词
（单数，第三）→气味
动词（复数，）→气味

图S23.2E1的部分DCG，修改为处理主语-动词编号/人协议。 22.2.

23.9部分DCG见图S23.2。我们包括人和数字注释，尽管英语实际上只区分动词协议的第三人称单数（除了动词be）。

23.10 一个解析捕获了"我能够钓鱼"的意思，另一个"我把鱼放在罐头里。〃
两者都有左分支NP→代词→I，其概率为0.16。

•第一个具有右分支VP→ModalVerb（0.2）与Modal→can（0.3）和Verb→fish（0.1），因此其先验概率为

$$0.16 \times 0.2 \times 0.3 \times 0.1 = 0.00096\,.$$

*第二个有右分支VP→VerbNP（0.8）与动词→can（0.1）和NP→名词→fish（0。6×0.3），所以它的先验概率是

$$0.16 \times 0.8 \times 0.1 \times 0.6 \times 0.3 = 0.002304\,.$$

As these are the only two parses, and the conditional probability of the string given the parse is 1, their conditional probabilities given the string are proportional to their priors and sum to 1: 0.294 and 0.706.

23.11 The rule for A is

$$A(n') \rightarrow aA(n)\ \{n' = \text{SUCCESSOR}(n)\}$$
$$A(1) \rightarrow a$$

The rules for B and C are similar.

$NP(case, number, Third) \rightarrow Name(number)$
$NP(case, Plural, Third) \rightarrow Noun(Plural)$
$NP(case, number, Third) \rightarrow Article(number)Noun(number)$
$Article(Singular) \rightarrow$ **a** | **an** | **the**
$Article(Plural) \rightarrow$ **the** | **some** | **many**

Figure S23.3 A partial DCG for $\mathcal{E}_1$, modified to handle article–noun agreement as in Ex. 22.3.

23.12 See Figure S23.3

23.13

a. Webster's New Collegiate Dictionary (9th edn.) lists multiple meaning for all these words except "multibillion" and "curtailing".

b. The attachment of all the propositional phrases is ambiguous, e.g. does "from . . . loans" attach to "struggling" or "recover"? Does "of money" attach to "depriving" or "companies"? The coordination of "and hiring" is also ambiguous; is it coordinated with "expansion" or with "curtailing" and "depriving" (using British punctuation).

c. The most clear-cut case is "healthy companies" as an example of HEALTH for IN A GOOD FINANCIAL STATE. Other possible metaphors include "Banks . . . recover" (same metaphor as "healthy"), "banks struggling" (PHYSICAL EFFORT for WORK), and "expansion" (SPATIAL VOLUME for AMOUNT OF ACTIVITY); in these cases, the line between metaphor and polysemy is vague.

23.14 This is a very difficult exercise—most readers have no idea how to answer the questions (except perhaps to remember that "too few" is better than "too many"). This is the whole point of the exercise, as we will see in exercise 23.14.

23.15 The main point of this exercise is to show that current translation software is far from perfect. The mistakes made are often amusing for students.

23.16 It's not true in general. With two phrases of length 1 which are inverted f_2, f_1, we have $d_1 = 0$ and $d_2 = 1 - 2 - 1 = -2$ which don't sum to zero.

23.17

由于这是唯一的两个解析，并且给定解析的字符串的条件概率为1，因此它们给定字符串的条件概率与其前序成正比，总和为1：0.294和0.706。

23.11 A的规则是

$$A(n') \rightarrow aA(n)\ \{n' = \text{SUCCESSOR}(n)\}$$
$$A(1) \rightarrow a$$

B和C的规则类似。

NP（case，number，Third）→Name（number）
NP（case，复数，第三）→名词（复数）
NP(case,number,Third)→冠词(number)名词(number)
文章(单数)→a|an/the
文章（复数）→the|some/many

Figure S23.3 E1的部分DCG，修改为处理冠词-名词协议，如Ex. 22.3.

23.12 见图S23.3

23.13

a. 韦伯斯特的新大学词典（9th edn。）列出除"数十亿"和"削减"之外的所有这些单词的多重含义。 b. 所有命题短语的附件都是模棱两可的，例如does"from。 . . 贷款"附加到"挣扎"或"恢复"？ "钱"是否附加到"剥夺"或"公司"？ "和雇用"的协调也是模棱两可的;它是与"扩张"协调，还是与"削减"和"剥夺"（使用英国标点符号）协调。 c. 最明显的例子是"健康的公司"，作为良好财务状况下健康的一个例子。 其他可能的隐喻包括"银行。 . . 恢复"（与"健康"相同的比喻），"银行挣扎"（工作的体力努力）和"扩张"（活动量的空间体积）;在这些情况下，隐喻和多义词之间的界限是模糊的。

23.14这是一个非常困难的练习--大多数读者不知道如何回答问题（除了也许要记住"太少"比"太多"好）。 这就是练习的全部内容，正如我们将在练习23.14中看到的那样。

23.15本练习的要点是表明当前的翻译软件远非完美。 所犯的错误对学生来说往往很有趣。

23.16一般情况下并非如此。 用两个长度为1的短语反转f2，f1，我们有d1=0和d2 = 1 −2 −1 = −2 总和不为零。

23.17

a. “I have never seen a better programming language” is easy for most people to see.

b. “John loves mary” seems to be prefered to “Mary loves John” (on Google, by a margin of 2240 to 499, and by a similar margin on a small sample of respondents), but both are of course acceptable.

c. This one is quite difficult. The first sentence of the second paragraph of Chapter 22 is “Communication is the intentional exchange of information brought about by the production and perception of signs drawn from a shared system of conventional signs.” However, this cannot be reliably recovered from the string of words given here. Code not shown for testing the probabilities of permutations.

d. This one is easy for students of US history, being the beginning of the second sentence of the Declaration of Independence: “We hold these truths to be self-evident, that all men are created equal . . .”

23.18

To solve questions like this more generally one can use the Viterbi algorithm. However, observe that the first two states must be onset, as onset is the only state which can output C_1 and C_2. Similarly the last two state must be end. The third state is either onset or mid, and the fourth and fifth are either mid or end. Having reduced to eight possibilities, we can exhaustively enumerate to find the most likely sequence and its probability.

First we compute the joint probabilities of the hidden states and output sequence:

$$\begin{aligned} P(1234466, OOOMMEE) &= 0.5 \times 0.2 \times 0.3 \times 0.7 \times 0.7 \times 0.5 \times 0.5 \\ &\quad \times 0.3 \times 0.3 . 7 \times 0.9 \times 0.1 \times 0.4 \\ &= 8.335 \times 10^{-6} \\ P(1234466, OOOMEEE) &= 5.292 \times 10^{-7} \\ P(1234466, OOOEMEE) &= 0 \\ P(1234466, OOOEEEE) &= 0 \\ P(1234466, OOMMMEE) &= 1.667 \times 10^{-5} \\ P(1234466, OOMMEEE) &= 1.058 \times 10^{-6} \\ P(1234466, OOMEMEE) &= 0 \\ P(1234466, OOMEEEE) &= 6.720 \times 10^{-8} \end{aligned}$$

We find the most likely sequence was O, O, M, M, M, E, E. Normalizing, we find this has probability 0.6253.

23.19 Now we can answer the difficult questions of 22.7:

- The steps are sorting the clothes into piles (e.g., white vs. colored); going to the washing machine (optional); taking the clothes out and sorting into piles (e.g., socks versus shirts); putting the piles away in the closet or bureau.
- The actual running of the washing machine is never explicitly mentioned, so that is one possible answer. One could also say that drying the clothes is a missing step.
- The material is clothes and perhaps other washables.

a. "我从未见过更好的编程语言"对大多数人来说很容易看到。 b. "约翰爱玛丽"似乎更喜欢"玛丽爱约翰"（在谷歌上，差距为2240到499，在一小部分受访者中也有类似的差距），但两者当然都是可以接受的。 c. 这一个是相当困难的。 第22章第二段的第一句话是"沟通是从常规标志的共享系统中得出的标志的生产和感知所带来的有意的信息交流。"但是，这不能从这里给出的单词串中可靠地恢复。 未显示用于测试排列概率的代码。 d. 这一条对美国历史的学生来说很容易，它是《独立宣言》第二句话的开头：“我们认为这些真理是不言而喻的，人人生而平等。 ..”

23.18

为了更普遍地解决这样的问题，可以使用Viterbi算法。 但是，观察前两个状态必须是起始状态，因为起始是唯一可以输出C1和C2的状态。 同样，最后两个状态必须结束。 第三种状态不是开始就是中期，第四种和第五种状态不是中期就是结束。 在减少到八种可能性后，我们可以穷举以找到最可能的序列及其概率。

首先，我们计算隐藏状态和输出序列的联合概率:

$$\begin{aligned} P(1234466, OOOMMEE) &= 0.5 \times 0.2 \times 0.3 \times 0.7 \times 0.7 \times 0.5 \times 0.5 \\ &\quad \times 0.3 \times 0.3 . 7 \times 0.9 \times 0.1 \times 0.4 \\ &= 8.335 \times 10^{-6} \\ P(1234466, OOOMEEE) &= 5.292 \times 10^{-7} \\ P(1234466, OOOEMEE) &= 0 \\ P(1234466, OOOEEEE) &= 0 \\ P(1234466, OOMMMEE) &= 1.667 \times 10^{-5} \\ P(1234466, OOMMEEE) &= 1.058 \times 10^{-6} \\ P(1234466, OOMEMEE) &= 0 \\ P(1234466, OOMEEEE) &= 6.720 \times 10^{-8} \end{aligned}$$

我们发现最可能的序列是O，O，M，M，M，E，E。 归一化，我们发现这有概率0.6253。

23.19 现在我们可以回答22.7的难题了:

*这些步骤是将衣服分类成堆（例如，白色与彩色）;去洗衣机（可选）;将衣服取出并分类成堆（例如，袜子与衬衫）;将堆放在壁橱或办公室里。 *洗衣机的实际运行从未明确提及，因此这是一个可能的答案。 人们也可以说，烘干衣服是一个缺失的步骤。 *材料是衣服，也许还有其他可洗的东西。

- Putting too many clothes together can cause some colors to run onto other clothes.
- It is better to do too few.
- So they won’t run; so they get thoroughly cleaned; so they don’t cause the machine to become unbalanced.

211

*把太多的衣服放在一起会导致一些颜色跑到其他衣服上。 *最好做得太少。 •所以他们不会运行;所以他们得到彻底清洁;所以他们不会导致机器变得不平衡。