14.2 Modify the algorithm for conversion to CNF from Chapter 13 to correctly handle rule probabilities. Make sure that the resulting CNF assigns the same total probability to each parse tree.

The three basic CNF transformation rules, and their corresponding probability calculations (shown in brackets following each rule):

- Replace $A \rightarrow B$ [$p_1$] rules with $A \rightarrow \beta_0 \ldots \beta_N$ [$p_1 \times p_2$] rules for each $B \rightarrow \beta_0 \ldots \beta_N$ [$p_2$] rule.

- Replace $A \rightarrow \beta_0 \ldots \beta_i b \beta_j \beta_N$ [$p_1$] rules (where $b$ is a terminal) with $A \rightarrow \beta_0 \ldots \beta_i B \beta_j \beta_N$ [$p_1$] and $B \rightarrow b$ [1.0] rules (where $B$ is a new symbol).

- Replace $A \rightarrow \beta_0 \ldots \beta_{N-2} \beta_{N-1} \beta_N$ [$p_1$] rules (where $N > 2$) with $A \rightarrow \beta_0 \ldots \beta_{N-2} B$ [$p_1$] and $B \rightarrow \beta_{N-1} \beta_N$ [1.0] rules (where $B$ is a new symbol).

14.3 Recall that Exercise 13.3 asked you to update the CKY algorithm to handle unit productions directly rather than converting them to CNF. Extend this change to probabilistic CKY.

```python
def prob_cky(grammar, words):
    ddict = collections.defaultdict
ddict(lambda: ddict(lambda: 0.0))
backs = ddict(lambda: {})

    # helpers for getting rules that produce the given symbols
    # and for getting heads of rules with probability > 0
    def get_rules(*symbols):
        for rule in grammar:
            if rule.symbols == symbols:
                yield rule

    def probs_positive(row, col):
        for head in probs[row, col]:
            if probs[row, col][head] > 0.0:
                yield head

    def add_unaries(row, col):
        seen = set()
        heads_todo = set(probs_positive(row, col))
        while heads_todo:
            head = heads_todo.pop()
            # add to the queue rules that could have generated
            # this symbol, that were not previously seen
            for rule in get_rules(head):
                if rule not in seen:
                    seen.add(rule)
                    heads_todo.add((rule, rule.head))

            # combine A -> B and B -> C rules and add the
            # new A -> C rule to the table
            prob = rule.prob * probs[row, col][head]
            if prob > probs[row, col][rule.head]:
                probs[row, col][rule.head] = prob
                back = None, head, None
                backs[row, col][rule.head] = back

        # for each word in the input, update the table cells in the
        # corresponding column, from bottom to top
```