```
tokens = ["3", "+", "5", "*", "(", "2", "+", "4", ")"]
position = 0
def match(token_type):
    global position
    if position < len(tokens) and tokens[position] == token_type:
        position += 1
        return True
    return False
def match_integer():
    global position
    if position < len(tokens) and tokens[position].isdigit():
        position += 1
        return True
    return False

def parse_expression():
    global position
    initial_pos = position

    if parse_term():
        while True:
            if match("+"):
                if not parse_term():
                    position = initial_pos
                    return False
            else:
                break
        return True
    position = initial_pos
    return False

def parse_term():
    global position
    initial_pos = position
    if parse_factor():
        while True:
            if match("*"):
                if not parse_factor():
                    position = initial_pos
                    return False
            else:
                break
        return True
    position = initial_pos
    return False

def parse_factor():
    global position
    initial_pos = position

    if match("("):
        if parse_expression() and match(")"):
            return True
    elif match_integer():
        return True

    position = initial_pos
    return False

result = parse_expression() and position == len(tokens)

if result:
    print("The expression is valid.")
else:
    print("The expression is invalid.")
```

```
The expression is valid.
```

```
grammar = {
    "S": [("NP", "VP")],
    "NP": [("Det", "N")],
    "VP": [("V", "NP")],
    "Det": [("the",)],
    "N": [("cat",), ("dog",)],
```

```
    "V": [("chased",), ("saw",)],
}

tokens = ["the", "cat", "chased", "the", "dog"]
n = len(tokens)

chart = [set() for _ in range(n + 1)]

chart[0].add(("S", (), ("NP", "VP"), 0))

pos = 0
while pos <= n:
    new_states = set()
    for state in chart[pos]:
        lhs, seen, unseen, origin = state

        if not unseen:
            for prev_state in chart[origin]:
                if prev_state[2] and prev_state[2][0] == lhs:
                    new_state = (prev_state[0], prev_state[1] + (lhs,), prev_state[2][1:], prev_state[3])
                    new_states.add(new_state)

        elif unseen[0] in grammar:
            for production in grammar[unseen[0]]:
                new_state = (unseen[0], (), production, pos)
                new_states.add(new_state)

        elif pos < n and unseen[0] == tokens[pos]:
            new_state = (lhs, seen + (unseen[0],), unseen[1:], origin)
            chart[pos + 1].add(new_state)

    chart[pos].update(new_states)

    pos += 1

valid = any(state == ("S", ("NP", "VP"), (), 0) for state in chart[n])

if valid:
    print("The sentence is valid.")
else:
    print("The sentence is invalid.")
```

```
The sentence is invalid.
```

```
import nltk
from nltk import CFG
from nltk.tree import Tree

grammar = CFG.fromstring("""
    S -> NP VP
    NP -> Det N | Det N PP
    VP -> V NP | VP PP
    PP -> P NP
    Det -> 'the' | 'a'
    N -> 'cat' | 'dog' | 'telescope' | 'park'
    V -> 'saw' | 'chased'
    P -> 'in' | 'with'
""")

sentence = "the cat saw the dog in the park".split()

parser = nltk.ChartParser(grammar)

for tree in parser.parse(sentence):
    print(tree)
    tree.pretty_print()
```

```
(S
  (NP (Det the) (N cat))
  (VP
    (VP (V saw) (NP (Det the) (N dog)))
    (PP (P in) (NP (Det the) (N park)))))
                  S
      ____________|_______
     |                    VP
     |            ________|_______
```

```
          |           VP              PP
          |        ___|___         ___|___
          NP       |       NP      |       NP
       ___|___     |    ___|___    |    ___|___
      Det      N   V  Det      N   P  Det      N
       |       |   |   |       |   |   |       |
      the     cat saw the     dog  in the     park

      (S
        (NP (Det the) (N cat))
        (VP
          (V saw)
          (NP (Det the) (N dog) (PP (P in) (NP (Det the) (N park))))))
                      S
          ____________|___
         |                VP
         |        ________|___
         |       |            NP
         |       |    ________|___
         |       |   |   |        PP
         |       |   |   |    ___|___
         NP      |   |   |   |       NP
      ___|___    |   |   |   |    ___|___
      Det      N   V  Det  N   P  Det      N
       |       |   |   |   |   |   |       |
      the     cat saw the dog  in the     park
```

```
import nltk
from nltk import CFG

grammar = CFG.fromstring("""
    S -> NP_SG VP_SG | NP_PL VP_PL

    NP_SG -> Det N_SG
    NP_PL -> Det N_PL

    VP_SG -> V_SG NP | V_SG
    VP_PL -> V_PL NP | V_PL

    Det -> 'the' | 'a'
    N_SG -> 'cat' | 'dog'
    N_PL -> 'cats' | 'dogs'
    V_SG -> 'chases' | 'sees'
    V_PL -> 'chase' | 'see'
""")

parser = nltk.ChartParser(grammar)

def check_agreement(sentence):
    tokens = sentence.split()
    try:
        parse_trees = list(parser.parse(tokens))
        if parse_trees:
            print("The sentence is grammatically correct with agreement.")
            for tree in parse_trees:
                print(tree)
        else:
            print("The sentence has a grammatical error (agreement issue).")
    except ValueError:
        print("The sentence has a grammatical error (agreement issue).")

sentence1 = "the cat chases the dog"
sentence2 = "the cats chase the dog"
sentence3 = "the cat chase the dogs"
sentence4 = "the dogs sees the cat"
print("Checking:", sentence1)
check_agreement(sentence1)
print("\nChecking:", sentence2)
check_agreement(sentence2)
print("\nChecking:", sentence3)
check_agreement(sentence3)
print("\nChecking:", sentence4)
check_agreement(sentence4)
```

```
Checking: the cat chases the dog
The sentence has a grammatical error (agreement issue).
```

```
Checking: the cats chase the dog
The sentence has a grammatical error (agreement issue).

Checking: the cat chase the dogs
The sentence has a grammatical error (agreement issue).

Checking: the dogs sees the cat
The sentence has a grammatical error (agreement issue).
```

```
import nltk
from nltk import PCFG
from nltk.parse import ViterbiParser

pcfg_grammar = PCFG.fromstring("""
    S -> NP VP [1.0]
    NP -> Det N [0.5] | Det N PP [0.5]
    VP -> V NP [0.5] | VP PP [0.5]
    PP -> P NP [1.0]
    Det -> 'the' [0.8] | 'a' [0.2]
    N -> 'cat' [0.5] | 'dog' [0.5]
    V -> 'chased' [0.6] | 'saw' [0.4]
    P -> 'with' [0.6] | 'in' [0.4]
""")

parser = ViterbiParser(pcfg_grammar)

sentence = "the cat chased the dog".split()

for tree in parser.parse(sentence):
    print(tree)
    tree.pretty_print()
```

```
(S
  (NP (Det the) (N cat))
  (VP (V chased) (NP (Det the) (N dog)))) (p=0.012)
              S
  ____________|_____
 |                  VP
 |              ____|___
 |             |        NP
 |             |      __|___
Det      N     V    Det      N
 |       |     |     |       |
the     cat chased the     dog
```