IMPLEMENTACIÓ DE L'ALGORISME CKY

Programació i Algorísmia Avanzada

Nadia Fernandez Dominguez i

Paula Justo Gili

Grau en Intel·ligència Artificial,

Facultat d'Informàtica de Barcelona, UPC

Diumenge 15 de Juny del 2025

ÍNDEX

1. Descripció de la tasca

En la pràctica es demana implementar l'algorisme CKY (Cocke–Kasami–Younger), utilitzat per analitzar si una paraula pot ser generada per una gramàtica lliure de context (CFG) expressada en forma normal de Chomsky (CNF). L'activitat consisteix a desenvolupar un programa en Python que rebi com a entrada una gramàtica i una paraula, i retorni un booleà indicant si la paraula forma part del llenguatge generat per la gramàtica.

El programa ha d'incloure una funció principal des d'on es pugui executar l'algorisme amb diferents conjunts de proves, i ha de contenir comentaris explicatius en les parts rellevants del codi. També es demana un conjunt de tests per validar el correcte funcionament de la implementació. A més, s'ha de redactar un document en PDF que reculli l'explicació del funcionament del codi, el format utilitzat per a representar les gramàtiques, i una valoració personal del procés d'aprenentatge.

De forma opcional, es pot implementar la transformació automàtica de CFG a CNF, així com una versió probabilística de l'algorisme CKY. Aquestes ampliacions només es valoraran si la part bàsica està completament funcional. Finalment, es destaca que el codi lliurat ha de ser totalment executable i sense errors, i que no s'accepta l'ús de biblioteques externes que resolguin directament la tasca proposada.

2. Implementacions

En aquesta pràctica, a més de desenvolupar l'algoritme bàsic del CKY per a la verificació de paraules segons una gramàtica en CNF, s'ha implementat també una funcionalitat addicional que permet transformar automàticament una gramàtica lliure de context a la seva forma normal de Chomsky. Paral·lelament, s'ha desenvolupat una versió probabilística de l'algorisme CKY, que calcula la probabilitat màxima d'una derivació d'una paraula segons una gramàtica amb regles probabilístiques. Per facilitar la utilització d'aquestes funcionalitats, s'ha creat una versió més interactiva i dinàmica del programa, que permet escollir de manera flexible entre aplicar l'algorisme CKY clàssic (amb transformació prèvia a CNF si cal) o bé la versió probabilística, tot plegat mitjançant una interfície més amigable per a l'usuari.

Els fitxers inclosos en el lliurament de la pràctica són els següents:

- CKY.py: conté la classe CKY, amb la implementació de l'algorisme en la seva versió bàsica (determinista).
- CKY_probabilistic.py: implementa la versió probabilística de l'algorisme CKY.
- gramatica.py: defineix la classe Gramatica per a gestionar gramàtiques en CNF.

- gramatica_probabilistic.py: conté una versió adaptada de la classe Gramatica per treballar amb gramàtiques amb probabilitats.
- main.py: permet executar l'algorisme CKY bàsic amb entrada de gramàtica i paraula.
- main_probabilistic.py: executa la versió probabilística de l'algorisme CKY.
- main_interactiu.py: proporciona una versió més interactiva que permet seleccionar entre les diferents funcionalitats (CKY normal amb transformació a CNF o CKY probabilístic).

2.1. Algoritme CKY

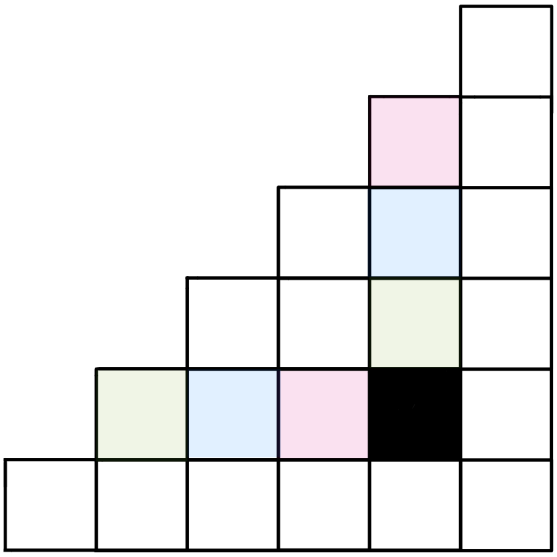
Dins de la implementació de l'algorisme CKY en la seva versió clàssica, s'han desenvolupat principalment dues classes: *CKY* i *Gramatica*, a més d'un fitxer d'execució, *main.py*, que permet provar automàticament l'algorisme sobre diferents conjunts de dades.

Classe CKY

Aquesta classe, definida al fitxer CKY.py, conté la lògica central de l'algorisme. Quan s'inicialitza, se li passa com a paràmetre una instància de *gramatica*. El seu mètode principal és ***pertany****(paraula)*, que rep com a entrada una paraula (una cadena de caràcters) i construeix una taula triangular per determinar si aquesta pot ser derivada pel símbol inicial de la gramàtica, applicant **programació dinàmica**. La taula és una llista de llistes de conjunts, on cada cel·la taula[i][j] conté els no terminals que poden derivar la subcadena que va de la posició i a la j de la paraula.

Primer s'inicialitza la diagonal principal (taula[i][i]) amb els símbols no terminals que generen directament cada símbol terminal de la paraula. Aquesta informació s'obté mitjançant el mètode trobar_cap de la classe Gramatica, que retorna tots els símbols que poden produir una determinada cadena (en aquest cas, un únic símbol terminal).

Després es construeix la resta de la taula de forma ascendent. Per cada cel·la concreta, busquem entre els elements de les cel·les de la mateixa fila però menor columna (*primera*) i els de la mateixa columna però major fila (*segona*). D'aquesta manera, anem buscant combinacions recorrent-les per veure si existeix algun element no terminal que faci la producció d'algun element de *primera* més algun element de *segona* (mitjançant *trobar_combinacions*) i s'afegeixen a la cel·la actual. A la següent imatge podem veure la combinació de cel·les que fa servir l'algortime de manera més visual per trobar els elements no terminals de cada cel·la.



Un cop la taula està completada, si el símbol inicial es troba a la posició taula[0][n-1], on n és la longitud de la paraula, significa que la paraula pot ser derivada de la gramàtica, i es retorna True. En cas contrari, es retorna False.

Classe Gramatica

Per dur a terme el procediment de l'algorisme, la classe CKY es recolza en la classe Gramatica, implementada al fitxer gramatica.py, que encapsula totes les funcionalitats relacionades amb la representació i manipulació dels CFG. Com a paràmetre per crear la classe, només trobem el símbol inicial de la gramàtica, el qual per defecte és S. Les regles es guarden en un diccionari on cada clau és un no terminal, i cada valor és una llista de produccions (representades com llistes de símbols terminals i/o no terminals). Per últim, al crear la classe s'inicialitza el paràmetre i el qual es farà servir més endevant per crear nous símbols no terminal.

Aquesta classe inclou els següents mètodes:

- **_existeix_cap**: Comprova si un símbol no terminal existeix com a cap.
- **_existeix_regla**: Comprova si una regla ja existeix.
- **__trobar_cos**: Obtenir les produccions associades a un símbol no terminal en el cas de que existeixi com a cap d'una regla.
- **trobar_cap**: Trobar tots els no terminals que poden generar una producció concreta i retornar-los en un set. És essencial per omplir la diagonal de la taula CKY.

- **_eliminar_regla**: Mirem si existeix la regla, i aleshores l'eliminem.
- **afegir_regla**: Primer, mirem si la regla existeix. Si no existeix, mirem si el cap ja existeix. En cas positiu, li afegim en forma de llista el nou cos dins de les produccions d'aquest cap. En cas negatiu, creem una nova parella clau valor al diccionari amb els elements de la regla.
- **trobar_combinacions**: Rep com a paràmetre dos sets a partir dels quals ha de buscar totes les regles que continguin un element del primer set i un element del segon per retornar-ne els caps. Element clau per completar la taula del CKY.
- **__str__**: Per escriure la gramàtica, primer imprimim 'Gramàtica:' i després totes les regles amb la forma *'$X \rightarrow Y\ Z \mid A\ B\ \ldots$'*.

Fitxer main.py

Finalment, el fitxer main.py actua com a script principal per executar l'algorisme sobre dades d'entrada. Aquesta és la estructura que ha de tenir el fitxer d'entrada:

S 3 *(nombre de regles a partir d'aquest no terminal)*
    AB BC ε *(produccions dels elements no terminals A i B, B i C, i la paraula buida)*
A 2
    BA a *(produccions dels elements no terminals B i A, i de l'element terminal a)*
B 2
    CC b
C 2
    AB a
paraules 3 ab a ε *(3 paraules per comprovar que pertanyen a la gramàtica i les paraules)*

(... següent gramàtica)

En aquest fitxer, assumim:

- Els símbols no terminals estan en majúscula.
- Els símbols terminals estan en minúscula.
- Tots els símbols estan formats únicament per un caràcter.
- El símbol inicial és *S*.

Aleshores, fem servir *pytokr* per llegir aquestes dades. Inicialment, creem una nova instància de la classe *Gramatica* i inicialitzem *num* a 1 per tenir un recompte de les gramàtiques que hi ha. Després, comencem un bucle sobre totes les paraules del fitxer input. Mentre no trobem paraules, vol dir que estem encara creant la gramàtica, per tant, afegim les regles que anem llegint amb el mètode *afegir_regla*. Quan ja hem afegit totes, trobem la paraula *paraules* i creem una instància de la classe

CKY amb la gramàtica creada. Per últim, comprovem si cadascuna de les que hi ha pertanyen a la gramàtica definida amb el mètode *pertany* i tornem a començar amb la següent gramàtica.

2.2. Transformació de qualsevol gramàtica CFG a CNF

Per afegir aquesta alternativa al nostre codi inicial, vam modificar només la clase *Gramatica* i el *main.py*, ja que un cop tenim la gramàtica en CNF, podem aplicar l'algoritme CKY de manera normal.

Classe Gramatica

En primer lloc, hem afegit un mètode anomenat **_es_cnf** a la classe que comprova si una gramàtica està en Forma Normal de Chomsky. Perquè retorni True, la gramàtica ha de complir les següents condicions:

- No pot estar el símbol inicial a la dreta de qualsevol regla.
- El cos de cada regla ha de contenir 2 símbols no terminals o un símbol terminal.
- La paraula buida només pot ser una producció del símbol inicial.

Així doncs, per fer aquestes comprovacions recorrem totes les regles de la gramàtica. Si la longitud de la producció és major a 2 retornem False. Si és 2, mirem si hi ha algun símbol terminal o el símbol inicial; en aquest cas retornem False. Per últim, si la longitud és 1, mirem si és un element no terminal o si és la paraula buida però el cap no es el símbol inicial; i en aquest cas també retornem False. Si hem recorregut totes les regles, retornem True.

Després, hem fet un altre mètode **convertir_cnf** que en el cas de que el CFG no hi estigui en CNF (ho comprovem amb el mètode *_es_cnf*), seguim els següents passos per convertir-lo.

1. Afegir un nou símbol inicial: Fem servir el mètode **__nou_inicial** que afegeix un nou símbol inicial que derivi en l'anterior inicial per evitar que ho tinguem a la dreta. En el cas de que la paraula buida es trobés en la gramàtica, la tornem afegir.

2. Eliminar les paraules buides (ε): Amb el mètode **__eliminar_epsilon**, trobem tots els caps que produeixin ε i que no siguin el símbol inicial i eliminem la regla. Després, revisem la resta de regles i si algun dels símbols que formen part del cos es troba entre els caps d'epsilon, afegim la mateixa regla on l'hem trobat però sense aquest símbol. En el cas de que la nova regla no tingui cap símbol, afegim ε i tornarem a aplicar el mètode.

3. Eliminar les regles unitàries: Per eliminar les regles on el cos és un únic símbol no terminal, fem servir el mètode **__eliminar_unitaries**. Per fer-ho, recorrem totes les regles i per cada cap, mirem si existeix alguna regla formada per un únic element no terminal. En cas

afirmatiu, afegim aquest símbol al set *eliminades* (per no repetir) i afegim totes les regles produïdes per aquest símbol al cap on ens trobem. Quan ja hem recorregut totes les regles del mateix cap, eliminem totes les regles formades per símbols del set *eliminades*.

4. Substituir elements terminals en regles no unitàries: Per fer-ho, hem fet servir el mètode **__substituir_terminals**, en el qual recorrem totes les regles buscant les quals el seu cos té una longitud major a 1 i hi ha un element terminal. En aquestes, substituim l'element terminal per *T_{element terminal en majúscula}* i afegim una nova regla amb aquest nou element que produeixi l'element terminal.

5. Descomposar regles llargues: Per fer que totes les regles amb elementss no terminals tinguin longitud dos, hem fet el mètode **__descompondre_llargues**. Per poder iterar sobre el diccionari i modificar-lo alhora, hem creat una copia per iterar sobre aquesta mentre modifiquem l'original. En primer lloc, busquem les regles de longitud major a 2. Quan les trobem, afegim la regla amb el primer símbol no terminal i un nou símbol *Z_{self.i}* (amb la i definida a l'init), afegim una nova regla on *Z_{self.i}* produeixi la resta de cos que no hem fet servir abans i eliminem la regla llarga inicial. Com pot passar que la longitud del cos sigui major a 3 i la nova regla per tant sigui major a 2, en aquest cas tornem a repetir el mètode *__descompondre_llargues*.

Fitxer main.py

En aquest fitxer, l'únic canvi que hem fet és cridar el mètode *convertir_cnf* un cop hem llegit la gramàtica sencera. Aquest mètode ja s'encarrega de comprovar si la gramàtica està en CNF amb *_es_cnf* i de fer tots els passos necessaris per aconseguir convertir-la. Després d'això, fem exactament el que feiem a la versió base aplicant l'algorisme CKY per comprovar si les paraules pertanyen o no.

2.3. Algoritme CKY Probabilístic

Per dur a terme aquesta nova variant, hem hagut de crear una nova classe anomenada *Gramatica_Probabilistic*. Per provar l'algorisme CKY, ara farem servir la classe *CKY_Probabilistic*. Per últim, s'aplicarà tot això automàticament a través del fitxer *main_probabilistic.py*, que és el que haurem d'executar a la terminal.

Gramatica_Probabilistic

Aquesta nova classe és una subclasse de *Gramatica*. Ara, el diccionari està definit de manera que les regles són llistes de tuples, on el primer element de la tupla es una llista amb els element no terminals que produeix la regla i el segon element és la probabilitat. Així doncs, hem hagut de sobreescriure tots

els mètodes els quals no eren compatibles amb aquesta nova versió del diccionari. Per fer-ho, a l'hora de recorre les regles, si volem els elements no terminals accedim al primer element de la tupla i si volem la probabilitat al segon.

També, hem creat el mètode **trobar_probabilitat**, el qual recorre totes les regles del cap corresponent fins trobar el cos. Aleshores, retorna la probabilitat d'aquella regla

A més a més, com donem per suposat que les gramàtiques que rebem a l'input estan en CNF, els mètodes *_es_cnf* i *convertir_cnf* no ens van falta.

CKY_Probabilistic

Ara, l'algorisme CKY treballa sobre la gramàtica *Gramatica_Probabilistic*. El desenvolupament d'aquest és igual que anteriorment amb la diferència que a cada cel·la trobem tuples de símbols no terminals i la probabilitat. Aquesta probabilitat es troba calculada al mètode de la gramàtica *trobar_combinacions*, el qual és la probabilitat del primer símbol multiplicada per la probabilitat del segon, tot això multiplicat per la probabilitat de la regla que li correspon. Aleshores, ara el mètode *pertany* retorna True o False més la probabilitat de pertànyer, que és la màxima que trobem a la posició *taula[0][n-1]*.

Fitxer main_probabilistic.py

Com que ara la gramàtica que hem de llegir és diferent, aquesta és la estructura que ha de tenir el fitxer d'entrada:

S 1 *(nombre de produccions de la regla)*
    ε *(paraula buida)* 0.2 *(probabilitat)*
S 2
    B *(primer símbol no terminal)* A *(segon símbol no terminal)* 0.8 *(probabilitat)*
A 1
    a (símbol terminal) 1
B 1
    b 1
paraules 3 ba a ε *(3 paraules per comprovar que pertanyen a la gramàtica i les paraules)*

(... següent gramàtica)

En aquest fitxer, assumim:

1. Els símbols no terminals estan en majúscula.
2. Els símbols terminals estan en minúscula.
3. El símbol inicial és *S*.

A diferència d'abans, ara definim regla a regla i els símbols poden tenir més d'un caràcter.

A més d'això, les úniques diferències amb el fitxer *main.py* és que gramàtica és una instància de *Gramatica_Probabilistic,* per tant hem d'afegir les probabilitats cada cop que cridem *afegir_regla* i l'algorisme és una instància de *CKY_Probabilistic*. Per últim A l'hora d'imprimir si les paraules pertanyen o no, també escrivim la probabilitat.

2.4. Versió interactiva

Per tal de facilitar la interacció amb les diferents funcionalitats desenvolupades en aquesta pràctica, s'ha creat una versió interactiva mitjançant el fitxer *main_interactiu.py*. Aquest script permet a l'usuari escollir entre executar l'algorisme CKY clàssic o la seva versió probabilística. Un cop seleccionada l'opció, es guia pas a pas l'usuari, a través de la terminal, en la construcció d'una gramàtica, demanant-li els símbols no terminals i les produccions corresponents, així com les probabilitats en cas de fer servir la versió probabilística.

El funcionament del programa està dissenyat per ser intuïtiu i completament guiat. A l'inici, es sol·licita a l'usuari que esculli entre dues opcions:

1. Executar l'algorisme CKY bàsic.
2. Executar l'algorisme CKY probabilístic.

A continuació, es demana el nombre de símbols no terminals que tindrà la gramàtica. El primer símbol introduït serà el que es consideri com el símbol inicial. Per cada no terminal, l'usuari ha d'indicar el nombre de produccions i introduir-les una a una. En la versió probabilística, a més, cal especificar la probabilitat associada a cada producció. Si es vol afegir la paraula buida, s'ha d'introduir com a ε.

En el cas de la versió CKY bàsica, el programa comprova automàticament si la gramàtica està en forma normal de Chomsky (CNF) i, si no ho està, en fa la conversió de manera interactiva (s'espera a que l'usuari cliqui la tecla enter per fer la conversió). En canvi, per a la versió CKY probabilística, és requisit que l'usuari introdueixi directament una gramàtica que ja estigui en CNF, ja que el programa no implementa la transformació automàtica en aquest cas.

Un cop creada la gramàtica, l'usuari pot introduir paraules per verificar si pertanyen al llenguatge generat:

- Amb CKY bàsic, s'indica si la paraula pertany o no a la gramàtica.

- Amb CKY probabilístic, a més de verificar si pertany o no, es mostra la probabilitat de la derivació més probable.

Aquest procés es pot repetir per a tantes paraules com es vulgui, fins que l'usuari decideixi finalitzar.

En l'apartat 3 d'aquest informe es podrà veure amb més claredat el funcionament del programa mitjançant exemples concrets.

3. Resultats dels jocs de prova

ALGORITME CKY (PASSANT A CNF)

Per provar si ens funcionava aquesta versió, hem fet el joc de prova *joc.inp* que en conté molts exemples dels qual aquest és un d'ells:

Gramàtica:
S → ε | l | F S C | B D | a
C → E F | A t
D → b
E → B B
F → A D
B → c
A → t

La gramàtica NO està en CNF

Gramàtica:
S → l | B D | a | F C | F Z_1
C → E F | A T_T
D → b
E → B B
F → A D
B → c
A → t
SS → ε | l | B D | a | F C | F Z_2
T_T → t
Z_1 → S C
Z_2 → S C

La gramàtica està en CNF

La paraula tbcctb SI pertany a la gramàtica.
La paraula ε SI pertany a la gramàtica.
La paraula cb SI pertany a la gramàtica.
La paraula tbc NO pertany a la gramàtica.

Veiem com al principi no estaba en CNF, perquè trobem regles que no compleixen les restriccions, com per exemple: S → F S C. Un cop s'ha convertit la gramàtica, veiem que ara sí compleix tot i ara el símbol inicial és SS.

Aquestes són les derivacions de les paraules que el programa afirma que pertanyen a la gramàtica:

- *tbcctb*: SS → F C → A D C → t D C → t b C → t b E F → t b B B F → t b c B F → t b c c F → t b c c A D → t b c c t D → t b c c t j
- *ε*: SS → ε
- *cb*: SS → B D → c D → c b

ALGORITME CKY PROBABILÍSTIC

Per provar si ens funcionava aquesta versió, hem fet el joc de prova *joc_prob.inp* que en conté dos exemples dels qual aquest és un d'ells:

Gramàtica:
S → A B (0.2) | o (0.27) | l (0.06) | C B (0.26) | D E (0.2)
A → w (0.2) | E F (0.47) | z (0.07) | J H (0.26)
B → A F (0.23) | o (0.33) | a (0.1) | e (0.19) | g (0.14)
E → a (0.06) | D F (0.64) | G F (0.25) | i (0.05)
D → f (0.32) | e (0.18) | G C (0.5)
F → G I (0.27) | e (0.4) | D C (0.28) | y (0.04)
J → F H (1.0)
C → l (1.0)
G → e (1.0)
H → w (1.0)
I → w (1.0)

La paraula eley SI pertany a la gramàtica amb una probabilitat de 0.001.
La paraula leyewell SI pertany a la gramàtica amb una probabilitat de 1.0624068000000001e-05.
La paraula ε NO pertany a la gramàtica.

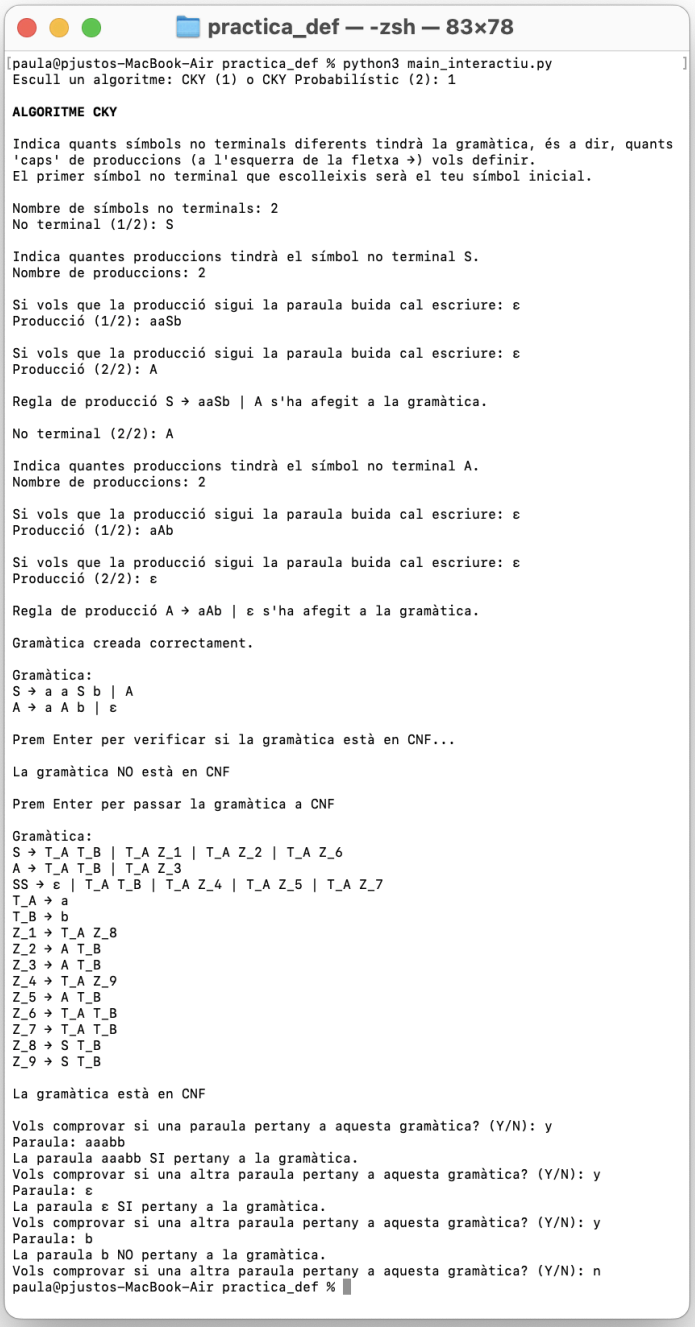
Veiem com les dues primeres paraules sí que pertanyen i la paraula buida no. Aquestes són les derivacions:

- *eley*: S → D E (0.2) → G C E (0.2 * 0.5 = 0.1) → e C E (0.1 * 1 = 0.1) → e l E (0.1 * 1 = 0.1) → e l G F (0.1 * 0.25 = 0.025) → e l e F (0.025 * 1 = 0.025) → e l e y (0.025 * 0.04 = 0.001)

- *leyewell*: S → C B (0.26) → l B (0.26 * 1 = 0.26) → l A F (0.26 * 0.23 = 0.0598) → l E F F (0.0598 * 0.47 = 0.028106) → l G F F F (0.028106 * 0.25 = 0.00703) → l e F F F (0.00703 * 1 = 0.00703) → l e y F F (0.00703 * 0.04 = 2.81e-4) → l e y G I F (2.81e-4 * 0.27 = 7.59e-5) → l e y e I F (7.59e-5 * 1 = 7.59e-5) → l e y e w F (7.59e-5 * 1 = 7.59e-5) → l e y e w D C (7.59e-5 * 0.28 = 2.12e-5) → l e y e w G C C (2.12e-5 * 0.5 = 1.06e-5)→ l e y e w e C C (1.06e-5 * 1 = 1.06e-5) → l e y e w e l C (1.06e-5 * 1 = 1.06e-5) → l e y e w e l l (1.06e-5 * 1 = 1.06e-5)

ALGORITME CKY INTERACTIU

Com a exemple de l'execució de la versió interactiva de l'algorisme CKY, s'ha realitzat el següent exemple:



Per tal de fer-lo servir des de la terminal, es crida a main_interactiu.py i, a partir d'aquí, es selecciona l'opció 1 per tal d'executar l'algorisme CKY clàssic. En aquest entorn, es pot introduir manualment la gramàtica que es vol utilitzar, sense necessitat que estigui en forma normal de Chomsky (CNF), ja que el programa s'encarrega de transformar-la automàticament si no ho està.

Per aquesta prova, s'ha construït una gramàtica amb 2 símbols no terminals: S (el símbol inicial) i A. Les produccions introduïdes han estat les següents:

S → aaSb | A
A → aAb | ε

Aquesta gramàtica no es troba inicialment en CNF, ja que algunes produccions tenen cossos amb més de dos símbols terminals o contenen la paraula buida (ε). Un cop introduïdes totes les regles, el programa ho detecta i transforma la gramàtica a forma normal de Chomsky, després que l'usuari hagi fet Enter. Un cop transformada, la gramàtica resulta ser:

S → T_A T_B | T_A Z_1 | T_A Z_2 | T_A Z_6
A → T_A T_B | T_A Z_3
SS → ε | T_A T_B | T_A Z_4 | T_A Z_5 | T_A Z_7
T_A → a
T_B → b
Z_1 → T_A Z_8
Z_2 → A T_B
Z_3 → A T_B
Z_4 → T_A Z_9
Z_5 → A T_B
Z_6 → T_A T_B
Z_7 → T_A T_B
Z_8 → S T_B
Z_9 → S T_B

Com es pot observar, s'han introduït noves variables (com Z_1, T_A, SS...) per tal d'adaptar les regles a CNF, descomponent produccions llargues, substituint terminals i tractant les produccions buides segons el procés explicat en l'apartat corresponent.

Un cop la gramàtica ha estat convertida, el programa pregunta si es vol comprovar si una paraula concreta pertany a la gramàtica. En aquest exemple, es proven les següents paraules:

- *aaabb*: El programa indica que sí que pertany a la gramàtica. Aquesta paraula pot ser derivada amb les regles S → aaSb → aaAb → aaaAbb → aaabb.
- *ε*: També s'indica que sí pertany a la gramàtica, ja que existeix explícitament la producció A → ε, que permet derivar la paraula buida.
- *b*: En aquest cas, el programa indica que no pertany a la gramàtica, ja que no hi ha cap combinació de produccions que permeti derivar únicament el símbol terminal b des del símbol inicial S.

ALGORITME CKY PROBABILÍSTIC INTERACTIU

Com a exemple de l'execució de la versió probabilística interactiva s'ha realitzat el següent exemple:

Per tal fer-lo servir des de la terminal es crida a main_interactiu.py i a partir d'aquí es selecciona l'opció 2, per tal d'executar l'algorisme en mode probabilístic. En aquest entorn, és on es pot introduir manualment la gramàtica que l'usuari desitja (en aquest cas, com es tracta de la versió probabilística l'usuari l'ha d'introduir directament amb forma normal de Chomsky, com ja s'ha comentat en l'apartat 2.4. d'aquest informe), juntament amb les probabilitats associades a cada producció.

Per aquesta prova s'ha construït gramàtica amb 11 símbols no terminals, sent el símbol inicial S. A continuació, s'han afegit les produccions i les probabilitats de cada producció per cada terminal:

- S → ε amb probabilitat 0.2
- S → l amb probabilitat 0.2
- S → A B amb probabilitat 0.37
- S → C D amb probabilitat 0.13
- S → i amb probabilitat 1.0
- B → E A amb probabilitat 1.0
- A → F D amb probabilitat 1.0
- E → C C amb probabilitat 1.0
- C → h amb probabilitat 1.0
- D → j amb probabilitat 1.0
- F → t amb probabilitat 1.0

Aquestes produccions compleixen les condicions de la CNF, ja que totes tenen cos de longitud 2 amb símbols no terminals o de longitud 1 amb un terminal.

Un cop definida tota la gramàtica, el programa mostra la seva representació completa. A continuació es demana a l'usuari si desitja o no comprovar si alguna paraula pertany o no a la gramàtica. En aquest cas sí que es vol comprovar i es comproven els següents paraules.

La primera paraula provada és *ε* (la paraula buida), la qual sí que pertany a la gramàtica (S → ε) i té una probabilitat de 0.2, tal com indica el programa.

A continuació, es prova la paraula *hj*. Aquesta paraula també és acceptada per la gramàtica amb una probabilitat de 0.13, tal i com mostra el programa. Aquesta derivació es pot fer a partir de la producció S → C D (0.13) → h D (0.13 * 1.0 = 0.13) → h j (0.13 * 1.0 = 0.13)

Finalment, s'ha provat una paraula que no pertany a la gramàtica: *tjhi*. El programa bàsicament el que fa és indicar que aquesta no pertany a la gramàtica, ja que no hi ha cap seqüència de produccions que permet derivar aquesta paraula des del símbol inicial S.

4. Conclusions i valoració personal del aprenentatge

El desenvolupament d'aquesta pràctica ha estat una oportunitat per consolidar els coneixements adquirits sobre programació dinàmica i anàlisi sintàctic aplicats a gramàtiques lliures de context. Implementar l'algorisme CKY, tant en la seva versió bàsica com en la probabilística, ha representat un repte interessant que ha ajudat a entendre millor com funcionen els reconeixedors sintàctics i quines estructures de dades cal utilitzar.

Afegir la transformació automàtica de CFG a CNF ha sigut especialment útil per veure de manera pràctica les restriccions formals de l'algorisme i com adaptar-hi qualsevol gramàtica. Aquesta part ens ha fet ser molt meticuloses amb la manipulació de les regles, especialment quan hem hagut de tractar produccions buides o unitàries.

La versió probabilística ens ha permès aprofundir en una perspectiva més realista del reconeixement de paraules, afegint-hi una dimensió quantitativa amb el càlcul de probabilitats. Això ens ha ajudat a veure aplicacions més pròximes al processament del llenguatge natural i ens ha servit per aplicar també conceptes que hem treballat a l'assignatura de Processament del Llenguatge Humà (PLH).

A més de fer la part obligatòria requerida, hem completat les dues extensions proposades i hem desenvolupat una versió interactiva del programa que fa més fàcil provar-lo i experimentar amb diferents gramàtiques i paraules.

Aquesta pràctica ens ha permès aplicar coneixements de diferents assignatures i aprofundir en tècniques de programació avançada, així com entendre millor la teoria que hi ha darrere.

Cal destacar també que el treball en grup ha estat molt positiu. La major part de la pràctica l'hem treballat juntes, compartint idees i prenent decisions conjuntament. Això ha permès que totes dues entenguéssim a fons cada part del codi i que poguéssim detectar i resoldre problemes amb més agilitat.