Pseudocode – Fonctions de Manipulation d'Automates

`lire_automate(fichier)`

1. Lire les lignes du fichier texte
2. Extraire `nb_symboles`, `nb_etats`, `etats_initiaux`, `etats_terminaux`, `nb_transitions`
3. Pour chaque ligne de transition :
   a. Valider les indices et les symboles
   b. Ajouter au dictionnaire `transitions`
4. Retourner un dictionnaire contenant :
   — `alphabet`, `etats`, `etats_initiaux`, `etats_terminaux`, `transitions`

`afficher_automate(automate)`

— Afficher les composants :
  — Alphabet
  — États
  — États initiaux
  — États terminaux
  — Transitions sous forme `"a -x-> b"`

`est_deterministe(automate)`

1. Vérifier qu'un seul état initial
2. Pour chaque transition :
   — Si un couple (état, symbole) mène vers plusieurs états → `return False`
3. Sinon, `return True`

`est_standard(automate)`

1. Vérifier qu'un seul état initial
2. S'assurer qu'aucune transition n'atteint cet état
3. Retourner `True` ou `False`

est_complet(automate)

1. Pour chaque (état, symbole) possible :
   — Vérifier que la transition existe
2. Retourner `True` si toutes sont présentes

verifier_automate(automate)

— Afficher si l'automate est :
   — Déterministe
   — Standard
   — Complet

standardiser(automate)

1. Si l'automate est déjà standard, `return`
2. Ajouter un nouvel état initial
3. Copier les transitions des anciens initiaux vers le nouveau
4. Retourner l'automate modifié

fermeture_epsilon(automate, etats)

1. Initialiser `pile = etats`
2. Tant que `pile` non vide :
   — Ajouter tous les états atteints via `'€'`
3. Retourner l'ensemble des états accessibles

contient_epsilon_transitions(automate)

— Retourner `True` si une transition a le symbole `'€'`

determiniser_automate_epsilon(automate)

1. Initialiser avec la fermeture de l'état initial
2. Pour chaque symbole `'€'` :
   — Calculer les états atteignables
   — Appliquer la fermeture
   — Créer les transitions dans un nouvel automate
3. Retourner l'AFD résultant

`determiniser_automate(automate)`

1. Créer état initial à partir des états initiaux
2. Explorer tous les symboles
3. Créer transitions et nouveaux états
4. Retourner l'AFD résultant

`completer_automate(automate)`

1. Si non déterministe, demander confirmation
2. Si complet, `return`
3. Ajouter un état puits
4. Ajouter toutes les transitions manquantes vers le puits
5. Retourner l'automate complété

`determiniser_et_completer(automate)`

1. Appliquer `determinisation` (avec ou sans)
2. Appliquer `completer_automate`
3. Retourner l'automate final

`creer_automate_complementaire(automate)`

1. S'assurer que l'automate est déterministe et complet
2. Inverser les états terminaux : tous les autres deviennent terminaux
3. Retourner l'automate complémentaire

`reconnaitre_mot(automate, mot)`

1. Partir de l'état initial
2. Pour chaque symbole :
   - Suivre la transition
   - Si transition manquante → rejet
3. Vérifier si l'état final est terminal → accepté ou rejeté

`reconnait_uniquement_mot_vide(automate)`

1. Vérifier :
   - Un seul état initial
   - C'est un état terminal
   - Il n'a aucune transition
2. Retourner `True` si toutes les conditions sont remplies

`minimiser_automate(automate)`

1. Si l'automate reconnaît uniquement le mot vide, retourner automate minimal
2. Déterminiser puis compléter
3. Partitionner les états (terminaux vs non-terminaux)
4. Raffiner les groupes par transitions
5. Recréer un nouvel automate en fusionnant les groupes
6. Retourner l'automate minimisé

`executer_tests_et_sauvegarder_traces(...)`

1. Lire automate
2. Vérifier ses propriétés
3. Standardiser
4. Déterminiser et compléter
5. Tester une série de mots
6. Créer le complémentaire
7. Sauvegarder toutes les traces dans un fichier