Support de Présentation

Automates finis et expressions rationnelles

Projet L2 - EFREI 2023

Jacques Soghomonyan
Antoine Ribot
Adrien Pouyat
Nicolas Chalumeau

Sommaire

E/S	État	a	b
⇆	0	.	1, 2
←	1	0, 2	2
→	2	0	.

Nous allons utiliser l'automate représenté par la table de transitions ci-dessus, pour illustrer nos algorithmes. (Automate n°40)

0.1 Structures de données

> Ici nous expliquerons le fonctionnement interne de la structure de données de l'automate.

0.1.1 Classe, méthodes et surcharge

Étant donné le choix de **python** en tant que langage. Le projet serra inévitablement orienté objet. De ce présupposé, nous avons implémenté une classe : **Automata**.

La classe est construite ainsi:

- Attributs :
  - entrees : liste des états entrants
  - exits : liste des états terminaux
  - alphabet : liste de chaques lettres composant l'alphabet de l'automate
  - transitions : la table de transition (cf. 0.1.2)
- Méthodes :
  - publique :
    - get_info() → str :Informations de l'automate (standard, déterministe, complet, N°transitions, d'entées et de sorties)
    - is_e_nfa() → bool : Si l'automate a des ϵ transitions.
    - to_dot_format() → str :
    - is_standard :
    - get_standard :
    - is_complete :
    - get_complete :
    - is_determinate :
    - get_state_e_closure :
    - get_simplified :
    - get_determinized :
    - test_word :
    - get_minimized :
    - get_complementary :
  - pseudo privés :
    - __give_state_behaviour :
    - __fetch_transition :
    - __populate_from_file :
    - __different_transitions_dict :
    - __is_state_empty :
    - __get_states :
    - __get_e_determinized :
- Surchages :
  - str :

– repr :
– contains :
– getitem/ setitem :

0.1.2 Table de transitions

Ici nous avons utilisé l'objet *dict* de python.

Le dictionnaire sera donc de la forme :

```
{
    etat_1: {
        lettre_1: [etat_j, ...],
        ...
        lettre_n: [etat_p, ...],
        },

    . . .

    etat_n: {
        lettre_1: [etat_r, ...],
        ...
        lettre_n: [etat_s, ...],
        },
}
```

Pour l'automate 40 :

```
{
    '0': {
        'a': [ ],
        'b': ['1', '2']
    },
    '1': {
        'a': ['0', '2'],
        'b': ['2']
    },
    '2': {
        'a': ['0'],
        'b': [ ]
    },
}
```

Cela permet une rapidité d'exécution et une lisibilité claire du code.

0.2 Algorithmes

Algorithme 1: Complétion

Données: AF
Résultat: AFC

```
1  AFC ← AF;
2  ajouterEtat(AFC, 'P');
3  pour lettre dans alphabet faire
4  |  ajouterTransitionAEtat('P', lettre, 'P');
5  fin
6  pour etat dans AFC faire
7  |  pour lettre dans alphabet faire
8  |  |  si estVide(etat) alors
9  |  |  |  ajouterTransitionAEtat(etat, lettre, 'P');
10 |  |  fin
11 |  fin
12 fin
13 retourner AFC;
```

Algorithme 2: Standardisation

Données: AF
Résultat: AFS

```
1  AFS ← AF;
2  ajouterEtat(AFS, 'I');
3  pour etat dans entrées(AFS) faire
4  |  transitionsEtat ← trouverTransition(AFS, etat);
5  |  pour lettre, etatReceveur dans transitionsEtat faire
6  |  |  ajouterTransitionAEtat('I', lettre, etatReceveur);
7  |  fin
8  fin
9  effacerEntrées(AFS);
10 ajouterEntrée(AFS, 'I');
11 retourner AFS;
```

Algorithme 3: Determinisation

Données: AF
Résultat: AFDC

```
1  AFD ← nouveau Automate;
2  pour etat dans entrées(AF) faire
3  |  transitionsEtat ← trouverTransition(AFS, etat);
4  |  pour lettre, etatReceveur dans transitionsEtat faire
5  |  |  ajouterTransitionAEtat('I', lettre, etatReceveur);
6  |  fin
7  fin
8  retourner Compléter(AFD)
```