Stage en laboratoire de recherche

Mise en place d'une bibliothèque traitant des automates en *Rust* et en *Haskell*

E. HADDAG, T. RENAUX VERDIERE

Université de Rouen Normandie
UFR Sciences & Techniques, campus du Madrillet

22 juin 2024

Contexte du stage

Figure – Logo du laboratoire

- Composer des dix enseignants chercheurs et de deux personnels administratif
- Ces domaines :
  - Informatique Quantique
  - Théorie des langages
  - Combinatoire
  - Génie logiciel

Sujet du stage

Étude de la conversion d'expression rationnelle en automates de *Glushkov*. Production d'une bibliothèque *Haskell* et *Rust* de manipulation de cette conversion.

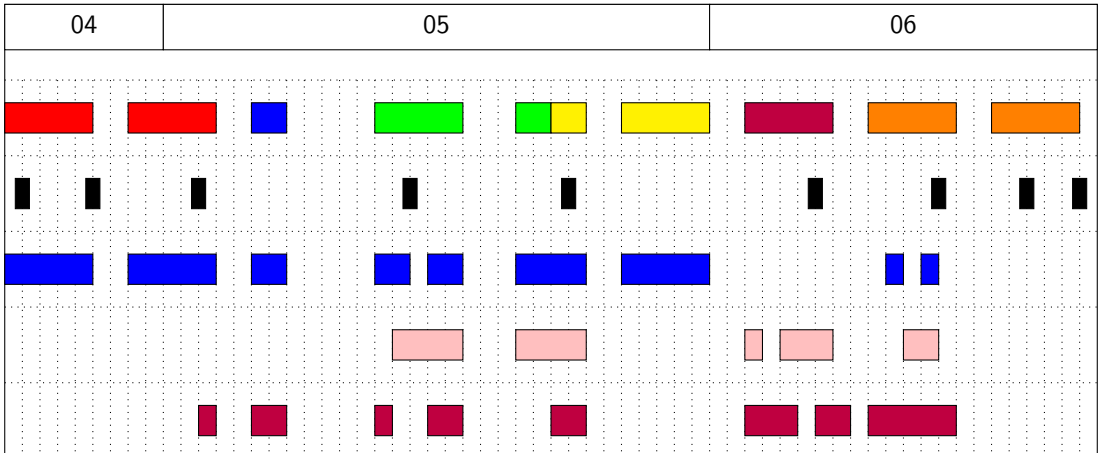


Figure – Diagramme de *Gantt*.

Mise en contexte

Expression régulière

Si E est une **expression régulière** sur Σ (un alphabet) elle peut être égale à :

- $E = \varnothing$
- $E = \varepsilon$
- $E = a \in \Sigma$
- $E = F + G$
- $E = F.G$
- $E = F^*$

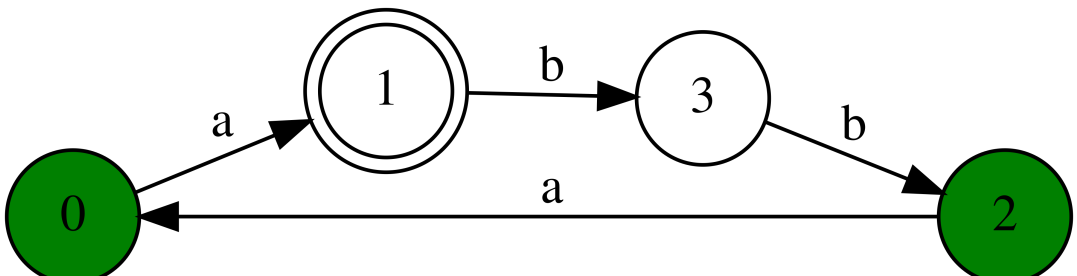
Avec F, G deux expressions régulières sur l'alphabet Σ.

Examples

Exemples $E = (a + b)^*.\varepsilon + \varnothing$

Automate

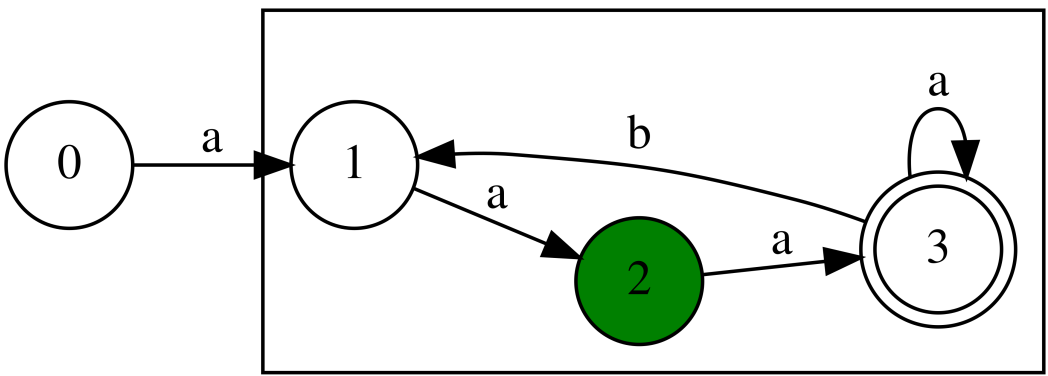
Un **automate** est un 5-uplet $(\Sigma, Q, I, F, \delta)$ qui peut être représenté graphiquement de la sorte :



Propriétés d'automates

Certaine propriété des automates définie par M.Caron et M.Ziadi :

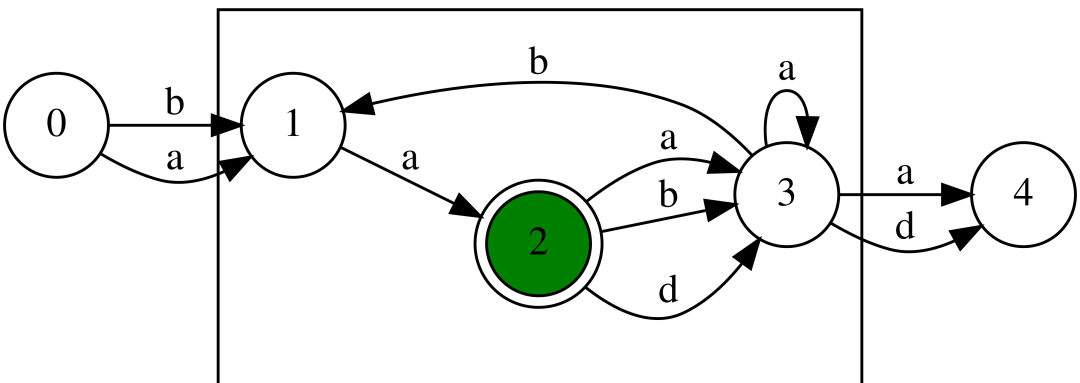
- une **orbite**
- une **orbite maximale**



Propriétés d'automates

Certaine propriété des automates définie par M.Caron et M.Ziadi :

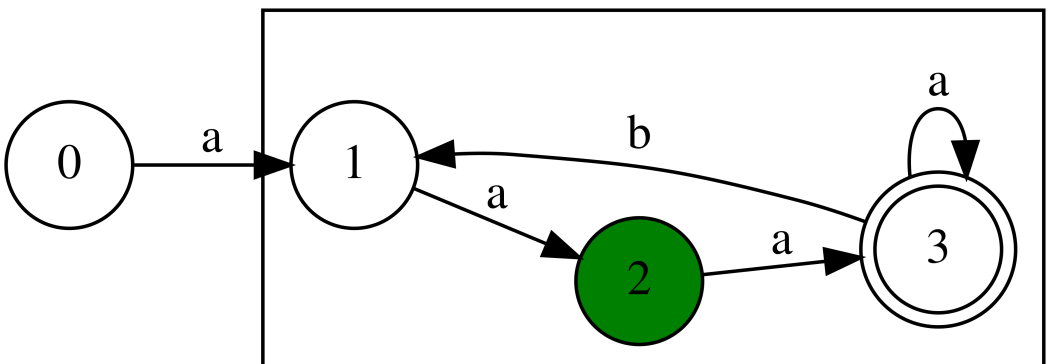
- les **portes d'une orbite**



Propriétés d'automates

Certaine propriété des automates définie par M.Caron et M.Ziadi :

- propriété de **stabilité** et **transversalité** (respectivement **fortement**).

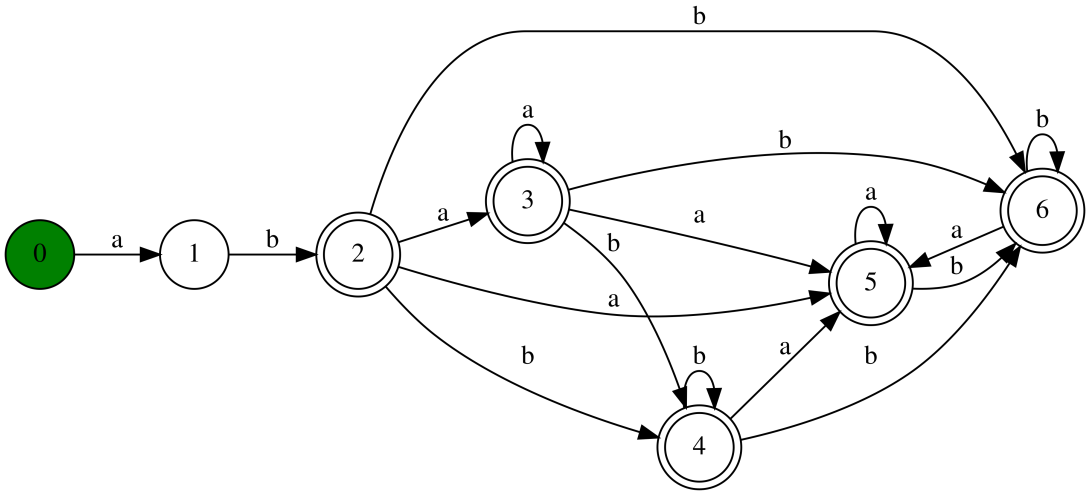


Glushkov et ces propriés

- Il existe un algorithme de transformation d'expressions régulières en automate qui est celui de *Glushkov*
- Toutes les orbites maximales des automates de Glushkov sont fortement stables et transversal

Automate

Automate de l'expression régulière $(a.b).a*.b*.(a+b)*$:



Implémentation

- Langage proche de la machine, au même niveau que le *C*
- Crée en 2006 par un employer de chez Mozilla qui a repris le projet après.
- Avec des systèmes de sécurisation du code
- Utilisé dans des domaines tels que :
  - Développement de Systèmes
  - Développement d'application
  - Dans des appareils embarqués

```rust
#[derive(Clone, Debug, PartialEq)]
pub enum RegExp<T> {
    Epsilon,
    Symbol(T),
    Repeat(Box<RegExp<T>>),
    Concat(Box<RegExp<T>>, Box<RegExp<T>>),
    Or(Box<RegExp<T>>, Box<RegExp<T>>),
}
```

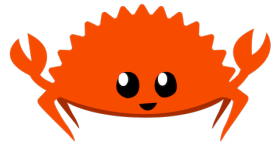
Figure – Définition du type `RegExp`, représentant une expression rationnelle.

```rust
#[derive(Debug)]
pub struct State<'a, T, V>
where
    T: Eq + Hash,
{
    value: V,
    previous: HashMap<T, HashSet<RefState<'a, T, V>>>,
    follow: HashMap<T, HashSet<RefState<'a, T, V>>>,
}
```

Figure – Définition du type `State`, représentant un état.

```rust
#[derive(Debug)]
pub struct InnerAutomata<'a, T, V>
where
    T: Eq + Hash + Clone,
{
    states: HashSet<RefState<'a, T, V>>,
    inputs: HashSet<RefState<'a, T, V>>,
    outputs: HashSet<RefState<'a, T, V>>,
}
```

Figure – Définition du type `InnerAutomata`, représentant un automate.



```
{
    "states":[6,5,3,4,2,1,0],
    "inputs":[0],
    "outputs":[6,2,3,4],
    "follows":[
        [6,"a",5],
        [5,"b",6],
        [3,"b",4],
        [3,"a",3],
        [3,"a",5],
        [4,"a",5],
    ]
}
```

Figure – Représentation d'un automate au format *JSON*

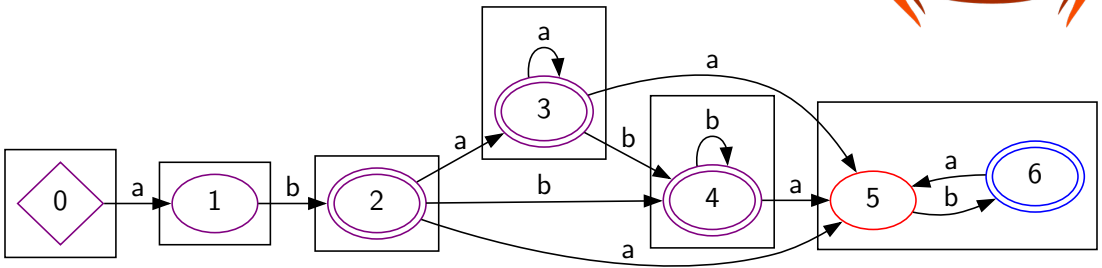


Figure – Représentation d'un automate au format *Dot*

Implémentation

- Langage fonctionnel **pur**
- Utilise des concepts de la **théorie des catégories** (**Monade**)
- Très forte abstraction qui mène à une sûreté d'exécution (Monade `IO`)
- Utilisé dans des domaines tels que :
  - La finance
  - Le militaire
  - La recherche

```haskell
data Exp a
    = Empty
    | Epsilon
    | Star (Exp a)
    | Plus (Exp a) (Exp a)
    | Point (Exp a) (Exp a)
    | Sym a
    deriving (Foldable, Functor, Traversable)
```

Figure – Définition du type `Exp`, représentant une expression rationnelle.

```haskell
class NFA nfa where
    type StateType nfa :: Type
    type TransitionType nfa :: Type
    -- Exemple de fonction de cette classe de type
    accept :: [TransitionType nfa] -> nfa -> Bool
    automatonToDot ::
        (Show (StateType nfa), Show (TransitionType nfa))
        => nfa
        -> DotGraph Gr.Node
```

Figure – Classe de type `NFA`.

```haskell
data NFAF state transition = NFAF
    { sigma   :: Set.Set transition
    , etats   :: Set.Set state
    , premier :: Set.Set state
    , final   :: Set.Set state
    , delta :: state -> transition -> Set.Set state
    }
```

Figure – Définition du type `NFAF`.

```haskell
data NFAG state transition = NFAG
    { sigma   :: Set.Set transition
    , etats   :: Map.Map state Int
    , premier :: Set.Set Int
    , final   :: Set.Set Int
    , graph   :: Gr.Gr state transition
    , lastN   :: Int
    }
```

Figure – Définition du type `NFAG`.

```
{
    "nodes": [0, 1, 2, 3],
    "first": [0, 2],
    "final": [1],
    "transitions": [
        [0, 1, "a"],
        [1, 3, "b"],
        [2, 0, "a"],
        [3, 2, "b"]]
}
```

Figure – Exemple de fichier *JSON* représentant un automate.

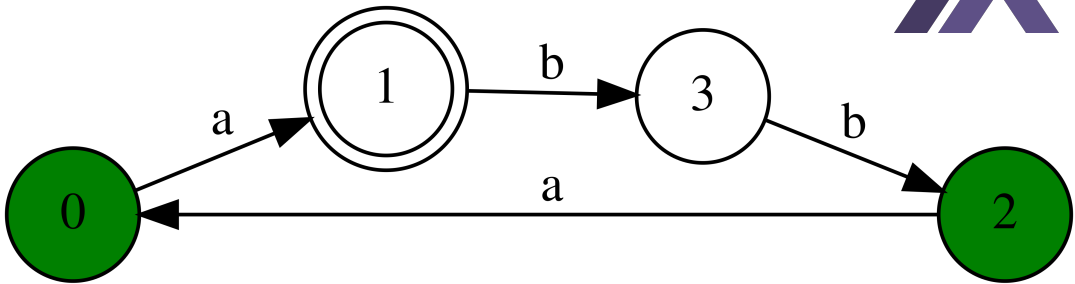


Figure – Représentation graphique de l'automate du fichier *JSON*.

```
$ stack test
Testing NG.NFAG implementation...
+++ OK, passed 100 tests.
+++ OK, passed 100 tests.
+++ OK, passed 100 tests.
+++ OK, passed 100 tests.
+++ OK, passed 100 tests.
+++ OK, passed 100 tests.
+++ OK, passed 100 tests.
Testing NF.NFAF implementation...
+++ OK, passed 100 tests.
+++ OK, passed 100 tests.
+++ OK, passed 100 tests.
+++ OK, passed 100 tests.
+++ OK, passed 100 tests.
+++ OK, passed 100 tests.
+++ OK, passed 100 tests.

Testing Glushkov properties...
+++ OK, passed 100 tests.
```

Figure – Résultat des tests par propriétés des types `NFAG` et `NFAF`.

Rapport du Benchmark

Démonstration des applications.

Bilan

Merci de nous avoir écoutés.