

# Stage en laboratoire de recherche

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

E. HADDAG, T. RENAUX VERDIRE

Université de Rouen Normandie  
UFR Science & Technique, campus du Madrillet

19 juin 2024

# Contexte du stage

---



Figure – Logo du laboratoire



Figure – Un des projets du laboratoire

## Sujet du stage

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

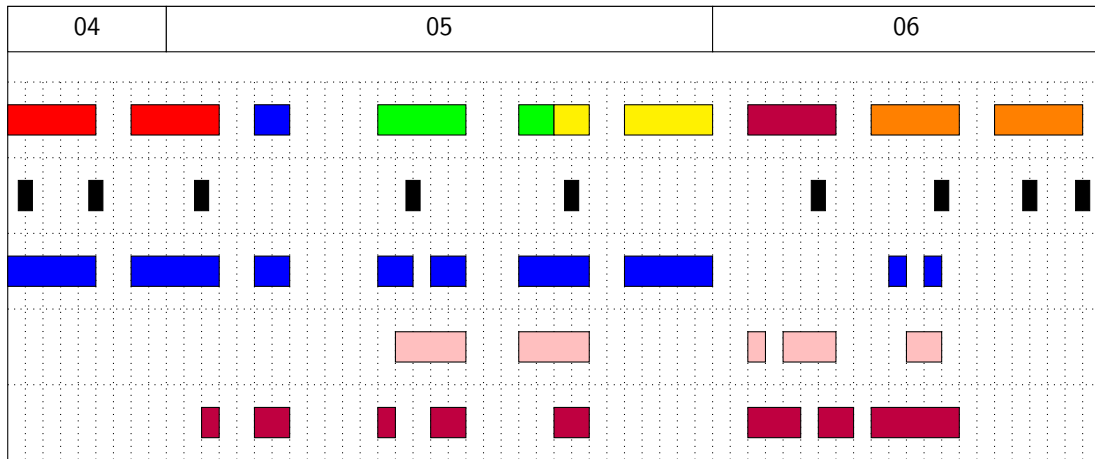
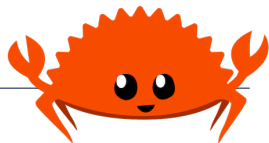


Figure – Diagramme de *Gantt*.



# Implémentation

---

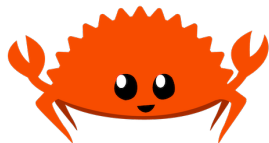


- Langage proche de la machine, au même niveau que le C
- Créé en 2006 par un employé de chez Mozilla qui a repris le projet après.
- Avec des systèmes de sécurisation du code
- Utilisé dans des domaines tels que :
  - Développement de Systèmes
  - Développement d'application
  - Dans des appareils embarqués



```
#[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.



```
#[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.





```
#[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],  
  ]  
}
```

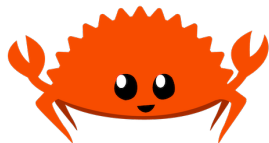


Figure – Représentation d'un automate au format *JSON*

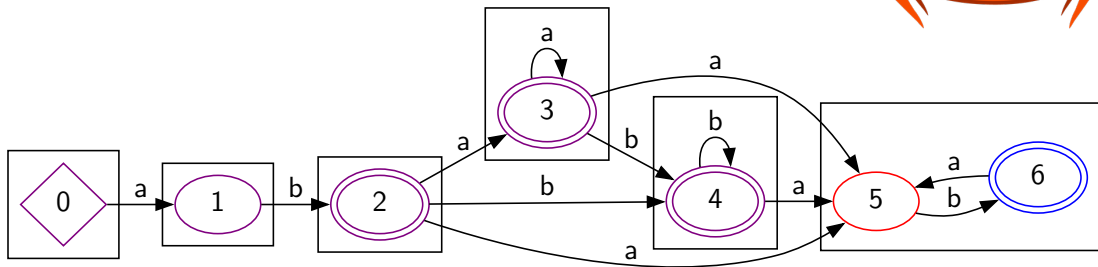
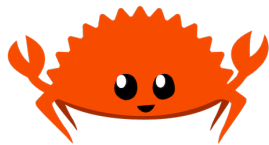


Figure – Représentation d'un automate au format *Dot*



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



```
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.



```
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**.



```
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**.



```
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.

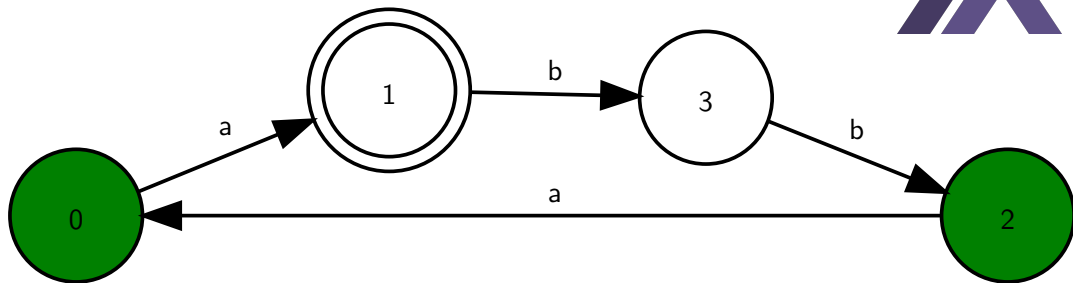


Figure – Représentation graphique de l'automate du fichier *JSON*.



---

```
$ 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 Gluskov properties...
+++ OK, passed 100 tests.
```

---

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



## Rapport du Benchmark



**Merci de nous avoir écoutés.**