



# Les formes narratives

propriétés formelles des récits : volume 1

Xavier Van de Woestyne  
**Touraine Tech 2018**

# Xavier Van de Woestyne

[twitter.com/vdwxv](https://twitter.com/vdwxv) [github.com/xvw](https://github.com/xvw) [xvw.github.io](https://xvw.github.io)

**OCaml, Haskell, Elm, Erlang, Elixir, Ruby, Io, PHP, Go**  
Nim, Racket, Ur/Web, **Java**, F#



- Formation techniques
- *API-first softwares* pour l'industrie du *Retail*
- **Elixir** et **Elm**
- Pas une agence web ni une société de consultants



- Meetup Lillois
- Un tous les deux mois
- Ouvert à tout sujet technique
- entre 25 et 80 participants
- **20 Mars 2018 !**



« *Useless software with  
useful languages* »

## 2012, « L'excursionniste »

- Un jeu d'aventure « *text-based* » développé en F#
- Un maximum de contenu généré
  - Chaines de Markov
  - BSP Trees
  - Automates cellulaires
  - Un algorithme un peu aléatoire pour les villes
  - Un autre algorithme aléatoire pour la distribution des opposants
  - Encore un autre algorithme aléatoire pour les conversations
- Pas de quêtes...

# Première tentative

## le tâtonnement

- Ajouter des objets, des personnages clés et des monstres nommés
- Intégrer un système de faits et de pondération
- Localiser l'apparition des éléments dans des sous-zones
- Introduction d'un système de « fuel »

# Vers une approche dirigée

## les cadres narratifs

- Campbell/Vogler : « *voyage du héros* »
- L'Oulipo et le roman policier
- Les clichés du RPG
- ...

...

Johann Wolfgang von Goeth  
Ferdinand de Saussure

Claude Levi-Strauss

Vladimir Propp

Algirdas Julien Greimas

Roland Barthes

Claude Bremond

Paul Ricoeur

...

Folk  
Myth  
Clock  
Textual

Vladimir Propp

La morphologie du conte, une histoire de classification  
(1928)

# Vladimir Propp

La morphologie du conte, une histoire de classification  
(1928)

- Tentative de classier des contes du folklore russes
- Des catégories finies aux motifs

# Observons ces deux fragments de contes

Le roi donne un aigle à un brave. L'aigle emporte le brave dans un autre royaume.

Un vieillard donne un anneau au héros. L'anneau, passé au doigt enferme le héros dans un puit.

**α** Situation initiale  
**A** Méfait  
**K** Réparation du méfait  
**β** Absence  
**a** Manque  
↓ Retour du héros  
**Y** Interdiction  
**B** Méditation  
**Pr** Poursuite  
**δ** Transgression  
**C** Début de l'opposition  
**Rs** Secours  
**e** Interrogation

↑ Départ du héros  
**O** Arrivée incognito du héros  
**ξ** Demande de renseignement  
**D** Première fonction du donateur  
**L** Imposture du héros  
**η** Tricherie  
**E** Réaction du héros  
**M** Tâche difficile  
**θ** Complicité  
**F** Transmission  
**N** Accomplissement de la tâche  
**G** Transfert du héros  
**Q** Découverte du faux héros

**H** Combat Opposant VS héros  
**T** Transfiguration  
**I** Marque/Don  
**U** Châtiment  
**J** Victoire  
**W** Mariage, monté sur le trône

**Fragment de conte précédent**

**αDF(↑)**

- L'agresseur
- Le donateur
- La princesse (ou personnage recherché)
- L'auxiliaire
- Le mandateur
- Le héros
- Le faux héros

**α A B C ↑ D E F G** { **H J**  
**I** } **K ↓ Pr-Rs O L Q Ex T U W**  
**M N**

# Typage d'une séquence ordonnée

dans un langage avec des types algébriques

# Éléments syntaxiques liés à OCaml

```
type firstname = string
type ('a, 'b) mapper = ('a -> 'b) -> 'a list -> 'b list

type point = (int * int)
type human = {
  firstname : string
  ; lastname : string
}

type 'a option =
| Some of 'a
| None
```

```
module Hello :
sig

  type firstname = string
  type lastname = string
  type an_abstract_type

  val hello : firstname -> lastname -> unit

end = struct

  type firstname = string
  type lastname = string
  type an_abstract_type = float

  let hello_to firstname lastname =
    "Hello " ^ firstname ^ " " ^ lastname

  let hello firstname lastname =
    print_endline (hello_to firstname lastname)
end
```

```
type propp =  
  | Alpha of string  
  | A of string  
  | K of string  
  (* ... etc *)  
  | J of string  
  | W of string  
  
type tale = propp list
```

```
let _ =  
  first "Adventure step 1"  
  → middle "Adventure step 2"  
  → middle "Adventure step 3"  
  → last "End of the adventure (step 4)"
```

```
let _ =  
  first "Adventure step 1"  
  → middle "Adventure step 2"  
  → last "End of the adventure (step 4)"  
  → middle "Adventure step 3"
```

```
let _ =  
  first "Adventure step 1"  
  → last "last"
```

# Utilisation de « types fantômes »

```
module type Nat =  
sig  
  type (+'a) t  
  val nat    : 'a t → 'a t t  
  val succ  : 'a t → 'a t t  
  val pred  : 'a t t → 'a t  
  
  val _0 : [`Zero] t  
  val _1 : [`Zero] t t  
  val _2 : [`Zero] t t t  
  val _3 : [`Zero] t t t t  
end
```

```
module type T = sig
  type (-'current, -'next) t
  val first : string → ([<`first], [<`middle]) t
  val middle : string → ([<`middle], [<`middle | `stop]) t
  val last : string → ([<`stop], [<`void]) t
  val (→) : ('a, 'b) t → ('b, 'c) t → ('b, 'c) t
end
```

# L'objectif

offrir des propriétés statiques a différents types de même « forme »

Et *quid* de la génération ?

# Travaux futures

- Libération du code source illustré et documenté
- Expérimenter l'utilisation de Coq pour des preuves plus fines
- Situation Calculus
- Ré-implémenter un jeu d'aventure ...

**Fin. Merci**