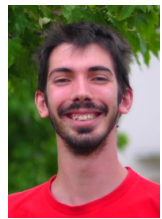
Vérification d'algorithmes OCaml 5 concurrents à grain fin en logique de séparation

Clément Allain

Curriculum vitae

- Master parisien de recherche en informatique (MPRI)
- Thèse à l'INRIA dirigée par François Pottier & Gabriel Scherer
- Postdoc à Aarhus University dirigé par Lars Birkedal

François
Pottier



Gabriel
Scherer

Lars
Birkedal

Vérification
d'algorithmes OCaml 5 concurrents à grain fin
en logique de séparation

$$\frac{P}{e}\quad\frac{}{v.\ Q}$$

$$\frac{\ell \mapsto v}{!\ell}\quad\frac{}{res.\ res = v * \ell \mapsto v}$$

Vérification
d'algorithmes OCaml 5 concurrents à grain fin
en logique de séparation

- **Fonctionnel** : fonctions de première classe, types algébriques.
- **Impératif** : références, enregistrements mutables.
- **Fortement typé** : typage $\Longrightarrow$ sûreté.
- **Ramasse-miettes** : gestion automatique de la mémoire.

Vérification d'algorithmes OCaml 5 concurrents à grain fin en logique de séparation

- **Parallélisme** : runtime multi-cœur, domaines, références atomiques.
- **Concurrence** : effets algébriques.
- **Écosystème naissant** : `Domainslib`, `Saturn`, `Eio` ...

Vérification
d'algorithmes OCaml 5 concurrents à grain fin
en logique de séparation

- **Gros grain** : verrous $\Longrightarrow$ sections critiques.
- **Grain fin** : primitives atomiques $\Longrightarrow$ interférences.

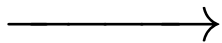
Zoo : un cadriciel pour la vérification de programmes OCaml

Contributions dans Zoo

Conclusion

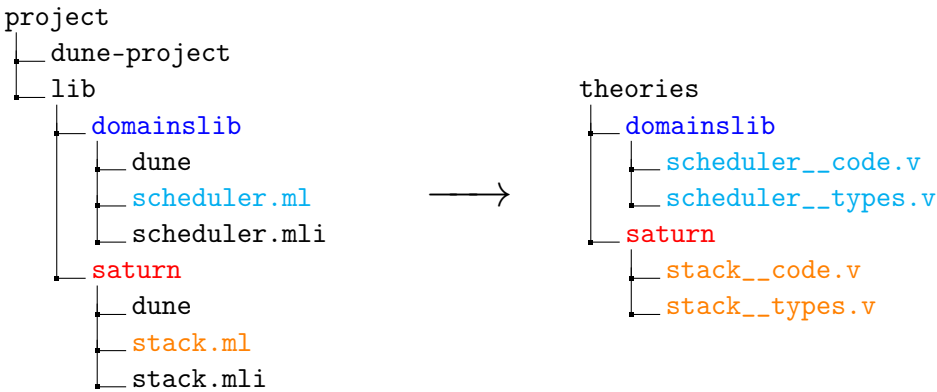
Zoo

`ocaml2zoo`



Zoo

Zoo en pratique




```
$ ocaml2zoo project theories
```

Fonctionnalités gérées

- Types algébriques
- Enregistrements
- Fonctions mutuellement récursives
- Références atomiques
- Champs atomiques
- Tableaux atomiques

- Égalité physique
- Égalité structurelle
- Constructeurs génératifs

- Variables prophétiques (linéarisation dépendant du futur)
- Diaframe (automatisation de base)

Zoo en pratique

```
let rec push t v =
  let old = Atomic.get t in
  let new_ = v :: old in
  if not (Atomic.compare_and_set t old new_) then (
    Domain.cpu_relax () ;
    push t v
  )
```

```
Definition stack_push : val :=
  rec: "stack_push" "t" "v" =>
    let: "old" := !"t" in
    let: "new" := ‘Cons( "v", "old" ) in
    if: ~ CAS "t" "old" "new" then (
      domain_cpu_relax () ;;
      "stack_push" "t" "v"
    ).
```

Zoo en pratique

```
Lemma stack_push_spec t ι v :
  <<<
    stack_inv t ι
  | ∀∀ vs, stack_model t vs
  >>>
    stack_push t v @ ↑ι
  <<<
    stack_model t (v :: vs)
  | RET (); True
  >>>.
Proof. ... Qed.
```

stack_push est *linéarisable*

Zoo en pratique

```
Lemma stack_push_spec t ι v :
  <<<
    stack_inv t ι
  | ∀∀ vs, stack_model t vs
  >>>
    stack_push t v @ ↑ι
  <<<
    stack_model t (v :: vs)
  | RET (); True
  >>>.
Proof. ... Qed.
```

stack_push est *linéarisable*

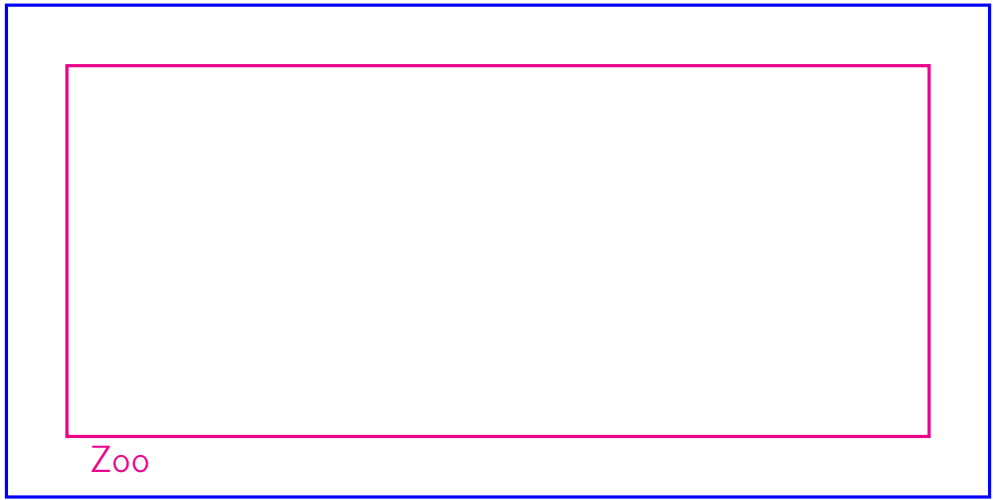
Zoo : un cadriciel pour la vérification de programmes OCaml

Contributions dans Zoo

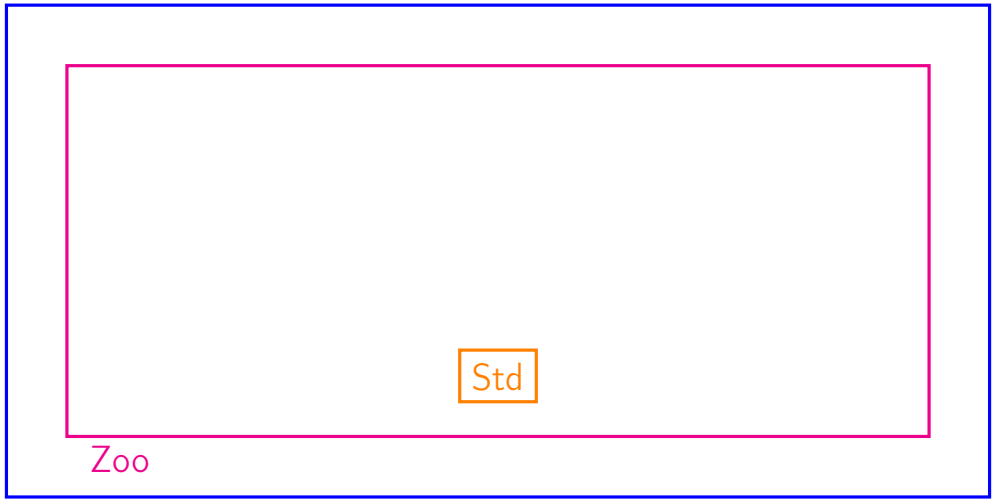
Conclusion

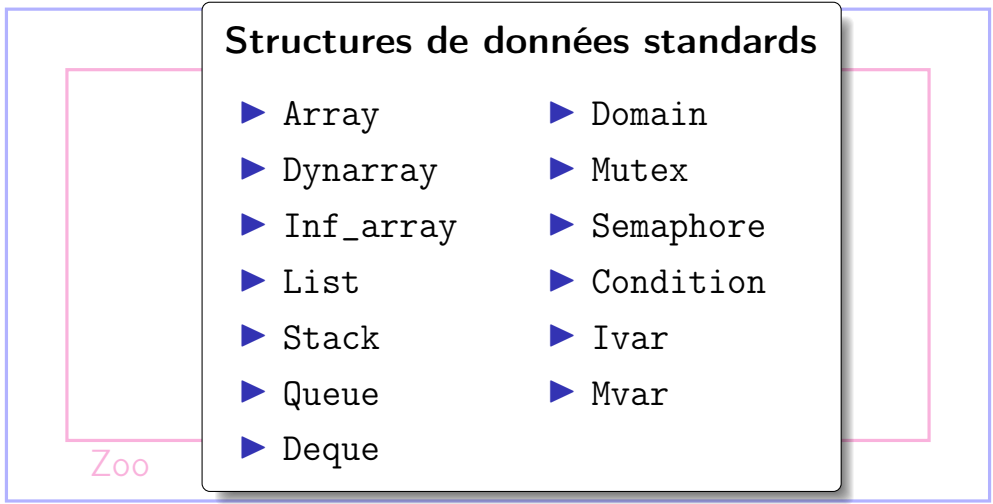
Contributions

Contributions

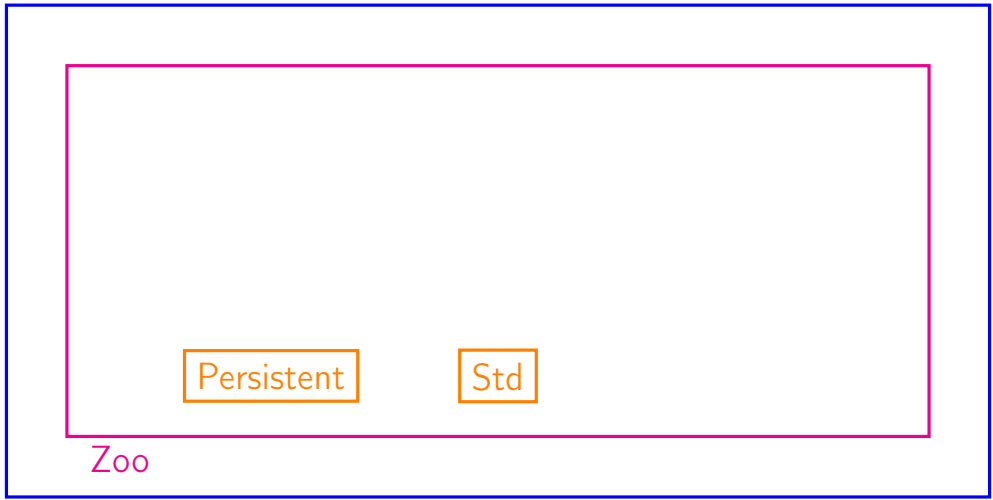


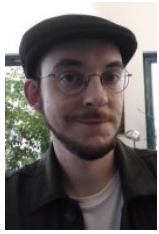
Contributions




Structures de données standards
Array
Dynarray
Inf_array
List
Stack
Queue
Deque
Domain
Mutex
Semaphore
Condition
Ivar
Mvar
Zoo
Rocq

Contributions





Basile Clément

Gabriel Scherer

Structures de données persistantes

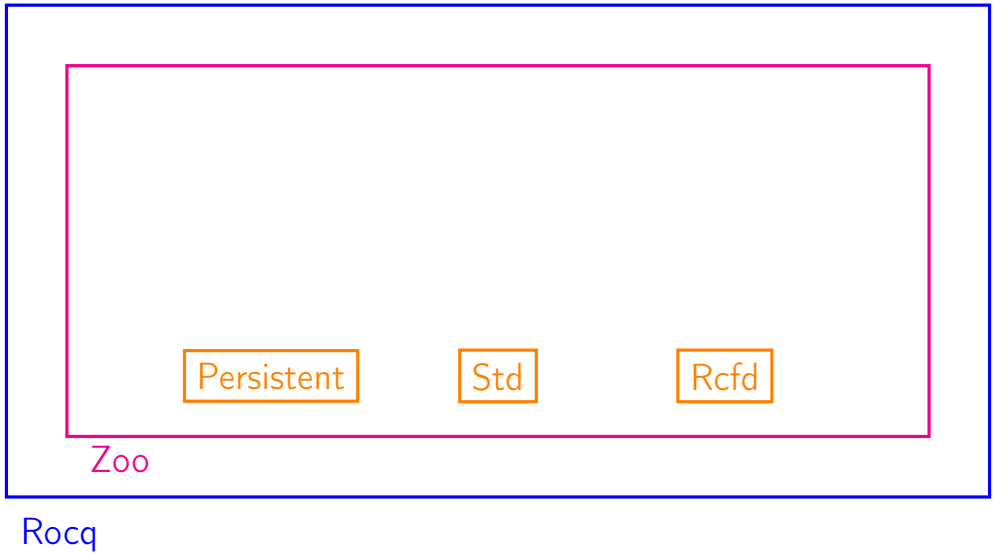
- Tableau persistant
- Store persistant
- Union-find persistant

Pe

Zoo

Rocq

Contributions



Rocq

Zoo

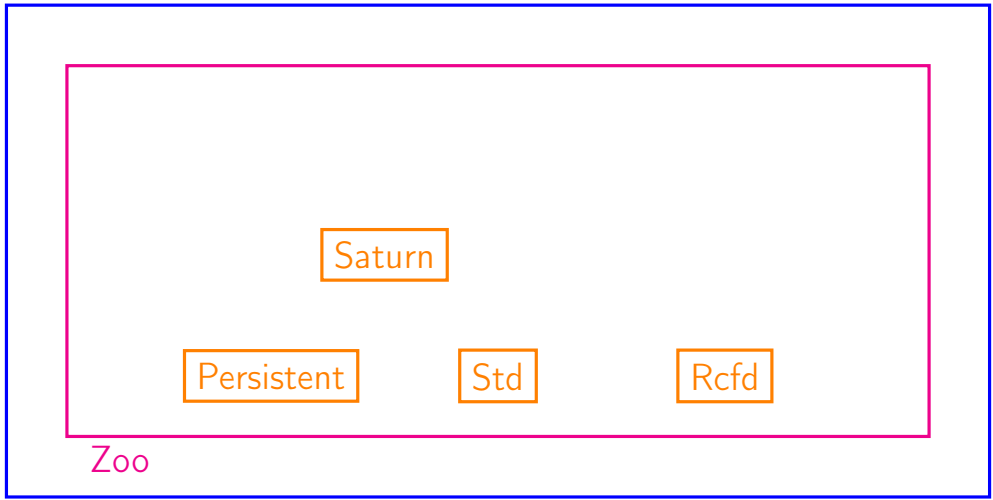
Per

Thomas Leonard

Descripteur de fichier concurrent

- Constructeurs génératifs
- Protocol concurrent subtil
- Deux régimes de propriété

Contributions



Vesa Karvonen **Carine Morel**

Structures de données lock-free

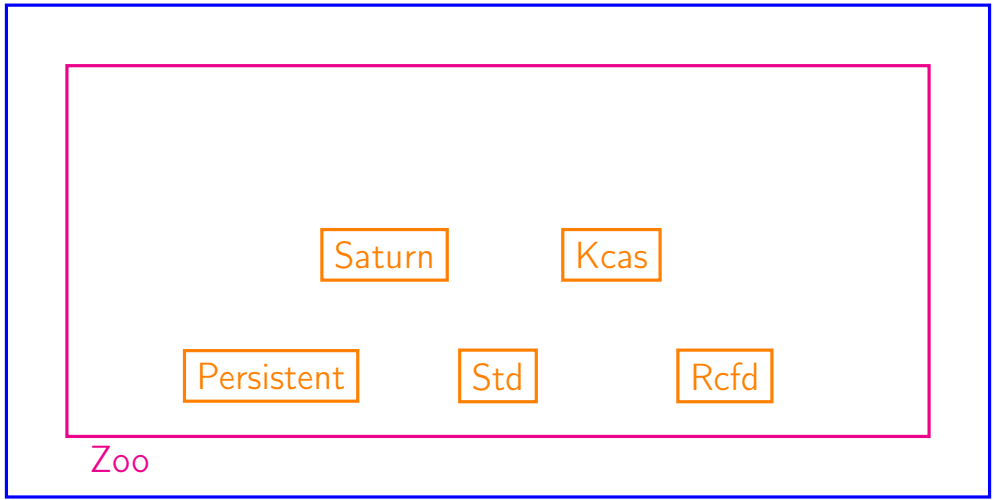
- Piles
- Queues basées sur des listes
- Queues basées sur des tableaux
- Queues basées sur des piles
- Deques de vol de tâches

Pe

Zoo

Rocq

Contributions



Vesa Karvonen

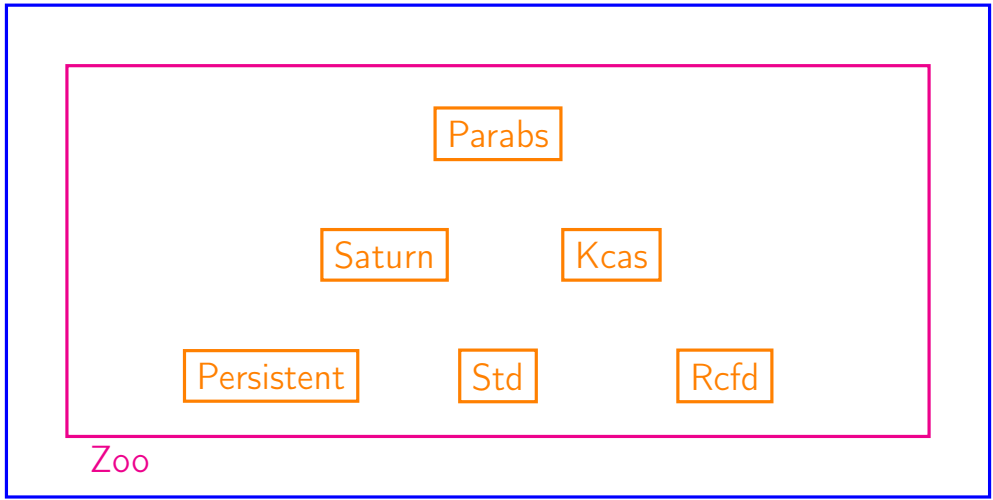
Algorithme compare-and-set multi-mot lock-free
(au cœur de la bibliothèque `Kcas`)

Pers

Zoo

Rocq

Contributions



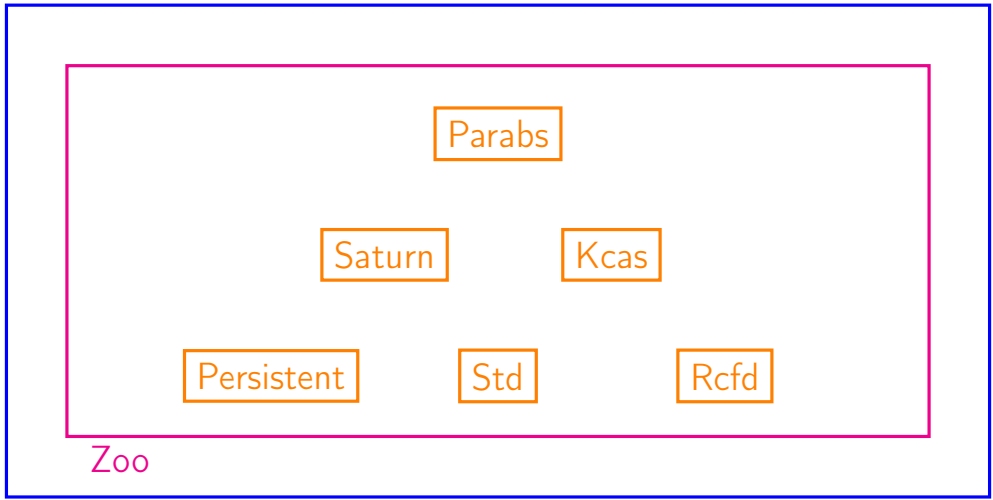
Abstractions parallèles

- Ordonnanceur par vol de tâches
- Futures
- Itérateurs parallèles
- Graphe de tâches (DAG-calcul)

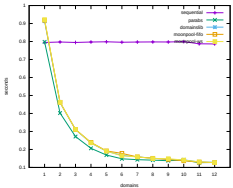
Zoo

Rocq

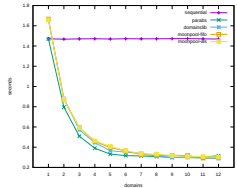
Contributions



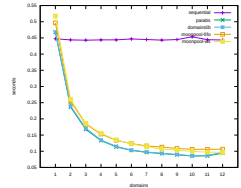
Benchmarks



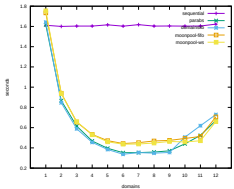
(a) `fibonacci`



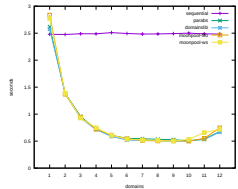
(b) `for_irregular`



(c) `iota`



(d) `lu`



(e) `matmul`

Les performances de `Parabs` sont égales ou supérieures à `Domainslib`.

Zoo : un cadriciel pour la vérification de programmes OCaml

Contributions dans Zoo

Conclusion

Thèse

Il est aujourd'hui possible de développer un écosystème de bibliothèques OCaml réalistes vérifiées (séquentielles et concurrentes) de façon modulaire.

Cette thèse pose les premières pierres de cet écosystème.

Travaux futurs

- **Fonctionnalités du langage**
  - Exceptions
  - Effets algébriques
  - Modules & foncteurs
- **Couplage avec vérification semi-automatique**
- **Mémoire faible**

Merci de votre attention !