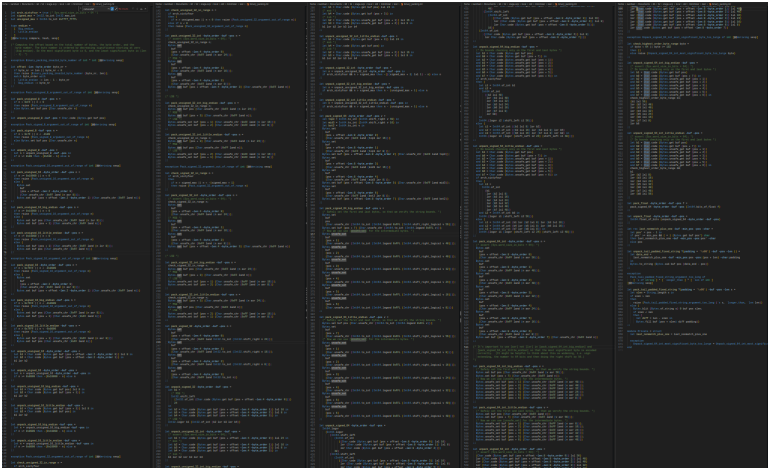
Chamelon: un minimiseur de programmes pour et en OCaml !

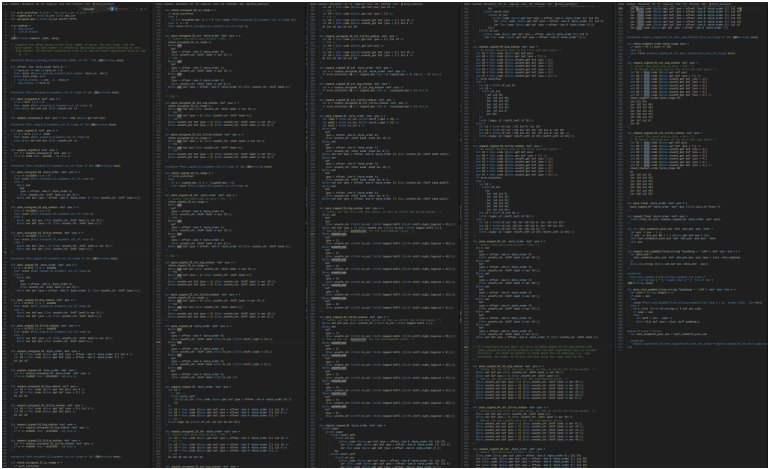
Milla Valnet[1,2,3],
Nathanaëlle Courant[3], Guillaume Bury[3], Pierre Chambart[3], Vincent Laviron[3]
[1]ENS Ulm, [2]Sorbonne Université, [3]OCamlPro

JFLA 2024

Motivation



Motivation



```
Misc.fatal_error
```

Motivation : pour les compilateurs, mais pas seulement !

- Pour les développeurs d'un compilateur OCaml

Motivation : pour les compilateurs, mais pas seulement !

- Pour les développeurs d'un compilateur OCaml
- Pour faire des bug reports sur du code confidentiel

Motivation : pour les compilateurs, mais pas seulement !

- Pour les développeurs d'un compilateur OCaml
- Pour faire des bug reports sur du code confidentiel
- Mais aussi adaptable pour d'autres outils travaillant sur du code OCaml !

Structure

- Génération du `.cmt`
- Extraction de l'AST typé

Structure

- Génération du `.cmt`
- Extraction de l'AST typé
- 18 minimiseurs atomiques...

Structure

- Génération du `.cmt`
- Extraction de l'AST typé
- 18 minimiseurs atomiques...
- Chacun appliqué autant que possible par un itérateur...

Structure

- Génération du `.cmt`
- Extraction de l'AST typé
- 18 minimiseurs atomiques...
- Chacun appliqué autant que possible par un itérateur...
- Le tout répété tant que des modifications sont effectuées !

Les minimiseurs 1/3

- Supprimer les définitions en commençant par la fin

Les minimiseurs 1/3

- Supprimer les définitions en commençant par la fin
- Remplacer les valeurs des définitions par des valeurs *dummy*

Les minimiseurs 1/3

- Supprimer les définitions en commençant par la fin
- Remplacer les valeurs des définitions par des valeurs *dummy*
- Supprimer le code mort

Les minimiseurs 1/3

- Supprimer les définitions en commençant par la fin
- Remplacer les valeurs des définitions par des valeurs *dummy*
- Supprimer le code mort
- Remplacer les expressions par des valeurs *dummy*

Les minimiseurs 1/3

- Supprimer les définitions en commençant par la fin
- Remplacer les valeurs des définitions par des valeurs *dummy*
- Supprimer le code mort
- Remplacer les expressions par des valeurs *dummy*
- Retirer les attributs des fonctions, modules, etc.

Les minimiseurs 1/3

- Supprimer les définitions en commençant par la fin
- Remplacer les valeurs des définitions par des valeurs *dummy*
- Supprimer le code mort
- Remplacer les expressions par des valeurs *dummy*
- Retirer les attributs des fonctions, modules, etc.
- Ajouter les attributs `[local never|always]` et `[inline never|always]`

Les minimiseurs 2/3

- Supprimer les constructeurs de types construits

Les minimiseurs 2/3

- Supprimer les constructeurs de types construits
- Supprimer les champs de types enregistrement

Les minimiseurs 2/3

- Supprimer les constructeurs de types construits
- Supprimer les champs de types enregistrement
- Supprimer les champs de constructeurs

Les minimiseurs 2/3

- Supprimer les constructeurs de types construits
- Supprimer les champs de types enregistrement
- Supprimer les champs de constructeurs
- Supprimer les expressions de type `unit`

Les minimiseurs 2/3

- Supprimer les constructeurs de types construits
- Supprimer les champs de types enregistrement
- Supprimer les champs de constructeurs
- Supprimer les expressions de type `unit`
- Simplifier les pattern matching

Les minimiseurs 2/3

- Supprimer les constructeurs de types construits
- Supprimer les champs de types enregistrement
- Supprimer les champs de constructeurs
- Supprimer les expressions de type `unit`
- Simplifier les pattern matching
- Simplifier les séquences

Les minimiseurs 3/3

- Inliner les fonctions

Les minimiseurs 3/3

- Inliner les fonctions
- Séquentialiser les applications de fonctions

Les minimiseurs 3/3

- Inliner les fonctions
- Séquentialiser les applications de fonctions
- Aplatir les modules

Les minimiseurs 3/3

- Inliner les fonctions
- Séquentialiser les applications de fonctions
- Aplatir les modules
- Simplifier les applications de fonction

Les minimiseurs 3/3

- Inliner les fonctions
- Séquentialiser les applications de fonctions
- Aplatir les modules
- Simplifier les applications de fonction
- Supprimer les `rec` inutilisés

Les minimiseurs 3/3

- Inliner les fonctions
- Séquentialiser les applications de fonctions
- Aplatir les modules
- Simplifier les applications de fonction
- Supprimer les `rec` inutilisés
- Supprimer les arguments inutilisés

Minimiseur unitaire

Étant donné un indice `i`, un minimiseur unitaire tente de minimiser le `i`-ème point de programme qu'il peut simplifier.

Minimiseur unitaire

Étant donné un indice `i`, un minimiseur unitaire tente de minimiser le `i`-ème point de programme qu'il peut simplifier.

Trois options :

- Cette simplification ne supprime pas l'erreur : on a minimisé le programme !

Minimiseur unitaire

Étant donné un indice `i`, un minimiseur unitaire tente de minimiser le `i`-ème point de programme qu'il peut simplifier.

Trois options :

- Cette simplification ne supprime pas l'erreur : on a minimisé le programme !
- Cette simplification supprime l'erreur : on ne souhaite pas l'appliquer.

Minimiseur unitaire

Étant donné un indice `i`, un minimiseur unitaire tente de minimiser le `i`-ème point de programme qu'il peut simplifier.

Trois options :

- Cette simplification ne supprime pas l'erreur : on a minimisé le programme !
- Cette simplification supprime l'erreur : on ne souhaite pas l'appliquer.
- L'indice est plus grand que celui du dernier emplacement qu'on peut modifier.

Type algébrique de retour

```
type 'a minimized_step_result =
  | New_state of 'a
  (* New (smaller) state that produces an error *)
  | Change_removes_error
  (* This change removes the error, but other might be possible *)
  | No_more_changes
  (* The last possible position for changes has been reached *)
```

La boucle principale

```
let minimize_basic (state : 'a)
  (f : 'a -> pos:int -> 'a minimized_step_result) : 'a * bool =
  let rec aux (state : 'a) (pos : int) (ever_changed : bool) =
    match f state ~pos with
    | New_state nstate -> aux nstate pos true
    | Change_removes_error -> aux state (pos + 1) ever_changed
    | No_more_changes -> (state, ever_changed)
  in
  aux state 0 false
```

La boucle principale

pos=0

La boucle principale

pos=0

La boucle principale

pos=0

La boucle principale

pos=0

La boucle principale

pos=0

La boucle principale

pos=0

La boucle principale

pos=0

La boucle principale

pos=0

La boucle principale

pos=1

La boucle principale

pos=1

La boucle principale

pos=1

La boucle principale

pos=1

La boucle principale

pos=1

La boucle principale

pos=1

La boucle principale

`pos=2`

La boucle principale

pos=2

La boucle principale

pos=2

La boucle principale

pos=2

La boucle principale

pos=2

La boucle principale

pos=2

La boucle principale

pos=2

La boucle principale

pos=3

La boucle principale

pos=3

La boucle principale

pos=4

La boucle principale

pos=4

La boucle principale

`pos=4`

La boucle principale

pos=4

Concrètement

- 18 minimiseurs

Concrètement

- 18 minimiseurs
- Basés sur Tast_mapper pour parcourir l'ast typé

Concrètement

- 18 minimiseurs
- Basés sur Tast_mapper pour parcourir l'ast typé
- Représentant 3000 lignes de code

Concrètement

- 18 minimiseurs
- Basés sur Tast_mapper pour parcourir l'ast typé
- Représentant 3000 lignes de code
- Déclinés en version compatible ocamlc ou flambda2

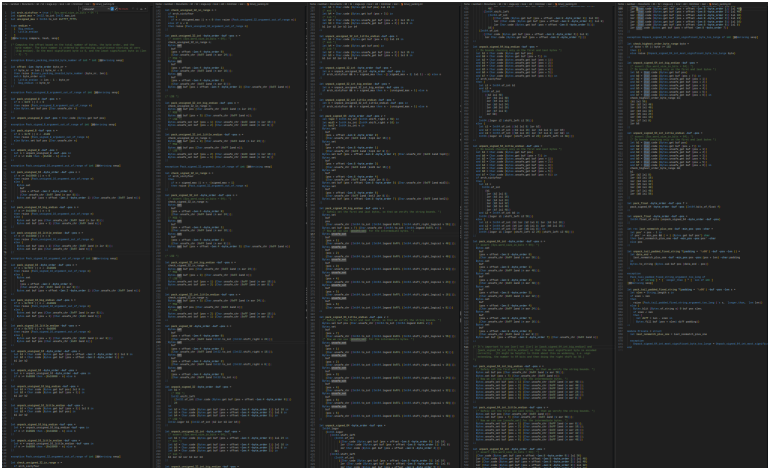
Concrètement

- 18 minimiseurs
- Basés sur Tast_mapper pour parcourir l'ast typé
- Représentant 3000 lignes de code
- Déclinés en version compatible ocamlc ou flambda2
- Déclinables à faible coût vers une version différente de l'AST via une librairie de compatibilité

Concrètement

- 18 minimiseurs
- Basés sur Tast_mapper pour parcourir l'ast typé
- Représentant 3000 lignes de code
- Déclinés en version compatible ocamlc ou flambda2
- Déclinables à faible coût vers une version différente de l'AST via une librairie de compatibilité
- Et donnant des résultats sur des exemples réels !

Démonstration



```
Misc.fatal_error
```

Démonstration

test > binary_packing_min.ml

```
1  let offset ~byte_order  byte_nr =
2    match byte_order with | `Little_endian -> 0 | `Big_endian -> byte_nr
3
4  let pack_unsigned_16 ~byte_order  =
5    __ignore__ ((offset ) ~byte_order 0);
6    __ignore__ ((__dummy2__ ()) ((offset ) ~byte_order 1));
7    __dummy2__ ()
```

```
Misc.fatal_error
```

D'un fichier seul à un projet complet: le multifichier

D'un fichier seul à un projet complet: le multifichier

- Suppression d'autant de fichiers que possible

D'un fichier seul à un projet complet: le multifichier

- Suppression d'autant de fichiers que possible
- Fusion d'autant de fichiers que possible

D'un fichier seul à un projet complet: le multifichier

- Suppression d'autant de fichiers que possible
- Fusion d'autant de fichiers que possible
- Représentation de l'ensemble des fichiers via une map

D'un fichier seul à un projet complet: le multifichier

- Suppression d'autant de fichiers que possible
- Fusion d'autant de fichiers que possible
- Représentation de l'ensemble des fichiers via une map
- Adaptation de tous les minimiseurs existants pour propager les modifications

D'un fichier seul à un projet complet: le multifichier

- Suppression d'autant de fichiers que possible
- Fusion d'autant de fichiers que possible
- Représentation de l'ensemble des fichiers via une map
- Adaptation de tous les minimiseurs existants pour propager les modifications

On fournit au minimiseur la liste des fichiers - `ocamldep` nous donne l'ordre de compilation.

On remplace notre méthode d'itération linéaire en dichotomique :

$\texttt{pos}=0 \wedge \texttt{lgth}=1$

On remplace notre méthode d'itération linéaire en dichotomique :

$\texttt{pos}=0 \wedge \texttt{lgth}=1$

On remplace notre méthode d'itération linéaire en dichotomique :

$\texttt{pos}=0 \wedge \texttt{lgth}=2$

On remplace notre méthode d'itération linéaire en dichotomique :

$\texttt{pos}=0 \wedge \texttt{lgth}=2$

On remplace notre méthode d'itération linéaire en dichotomique :

$\texttt{pos}{=}0 \wedge \texttt{lgth}{=}4$

On remplace notre méthode d'itération linéaire en dichotomique :

$\texttt{pos}{=}0 \wedge \texttt{lgth}{=}4$

On remplace notre méthode d'itération linéaire en dichotomique :

$\texttt{pos}{=}0 \wedge \texttt{lgth}{=}8$

On remplace notre méthode d'itération linéaire en dichotomique :

$\texttt{pos}{=}0 \wedge \texttt{lgth}{=}8$

On remplace notre méthode d'itération linéaire en dichotomique :

$\texttt{pos}{=}0 \wedge \texttt{lgth}{=}4$

On remplace notre méthode d'itération linéaire en dichotomique :

$\texttt{pos}{=}0 \wedge \texttt{lgth}{=}4$

On remplace notre méthode d'itération linéaire en dichotomique :

pos=0 $\wedge$ lgth=2

On remplace notre méthode d'itération linéaire en dichotomique :

$\texttt{pos}{=}0 \wedge \texttt{lgth}{=}2$

On remplace notre méthode d'itération linéaire en dichotomique :

$\texttt{pos}{=}0 \wedge \texttt{lgth}{=}2$

On remplace notre méthode d'itération linéaire en dichotomique :

$\texttt{pos}{=}0 \wedge \texttt{lgth}{=}1$

On remplace notre méthode d'itération linéaire en dichotomique :

$\texttt{pos}=0 \wedge \texttt{lgth}=1$

On remplace notre méthode d'itération linéaire en dichotomique :

$\texttt{pos}=0 \wedge \texttt{lgth}=1$

On remplace notre méthode d'itération linéaire en dichotomique :

$\texttt{pos}{=}1 \wedge \texttt{lgth}{=}1$

Optimisation

On remplace notre méthode d'itération linéaire en dichotomique :

$\texttt{pos}{=}1 \wedge \texttt{lgth}{=}1$

On remplace notre méthode d'itération linéaire en dichotomique :

$\texttt{pos}{=}1 \wedge \texttt{lgth}{=}2$

On remplace notre méthode d'itération linéaire en dichotomique :

pos=1 $\wedge$ lgth=2

On remplace notre méthode d'itération linéaire en dichotomique :

$\texttt{pos}{=}1 \wedge \texttt{lgth}{=}4$

On remplace notre méthode d'itération linéaire en dichotomique :

pos=1 $\wedge$ lgth=4

On remplace notre méthode d'itération linéaire en dichotomique :

pos=1 $\wedge$ lgth=2

On remplace notre méthode d'itération linéaire en dichotomique :

$\texttt{pos}{=}1 \wedge \texttt{lgth}{=}2$

On remplace notre méthode d'itération linéaire en dichotomique :

`pos=1` $\wedge$ `lgth=1`

On remplace notre méthode d'itération linéaire en dichotomique :

`pos=1` $\wedge$ `lgth=1`

On remplace notre méthode d'itération linéaire en dichotomique :

$\texttt{pos}{=}1 \wedge \texttt{lgth}{=}1$

On remplace notre méthode d'itération linéaire en dichotomique :

pos=2 $\wedge$ lgth=1

On remplace notre méthode d'itération linéaire en dichotomique :

pos=2 $\wedge$ lgth=1

On remplace notre méthode d'itération linéaire en dichotomique :

$\texttt{pos}{=}2 \wedge \texttt{lgth}{=}1$

Optimisation

On remplace notre méthode d'itération linéaire en dichotomique :

`pos`=2 $\wedge$ `lgth`=1

On gagne jusqu'à un facteur 10 pour un fichier de 4000 lignes !

Erreurs à runtime

Parfois, le plantage n'a pas lieu à la compilation, mais à l'exécution du code produit !

Erreurs à runtime

Parfois, le plantage n'a pas lieu à la compilation, mais à l'exécution du code produit !

$\Longrightarrow$ Restriction du minimiseur à des transformations safe

Erreurs à runtime

Parfois, le plantage n'a pas lieu à la compilation, mais à l'exécution du code produit !

$\Longrightarrow$ Restriction du minimiseur à des transformations safe

$\Longrightarrow$ Modification des valeurs *dummy*, qui utilisent une primitive unsafe : les expressions sont plutôt remplacées par des expressions simplifiées du même type !

Travaux futurs

- Rendre le minimiseur compatible avec un projet dune

Travaux futurs

- Rendre le minimiseur compatible avec un projet dune
- Plus de tests, sur plus de fichiers, pour en améliorer la précision, la robustesse, l'efficacité !

Résultats du stage

- Un minimiseur pour assister le développement d'un projet manipulant du code OCaml...

Résultats du stage

- Un minimiseur pour assister le développement d'un projet manipulant du code OCaml...
- Qui minimise de manière efficace des exemples réels...

Résultats du stage

- Un minimiseur pour assister le développement d'un projet manipulant du code OCaml...
- Qui minimise de manière efficace des exemples réels...
- Et qui fait gagner du temps à ses utilisateurs !

Merci pour votre attention !