LaFoSec: Étude de la sécurité intrinsèque des langages fonctionnels [1]

Partie III sur IV

Propositions d'évolution du langage OCaml
Conclusions de la Tranche Ferme

Damien Doligez, Christèle Faure, Thérèse Hardin, Manuel Maarek

JFLA - février 2013

1. Etude commanditée par l'Agence Nationale de la Sécurité des Systèmes d'Information

Introduction

Evolutions

Conclusion

Conclusion de la Tranche Ferme

Contacts

1 Introduction

2 Evolutions

3 Conclusion

4 Conclusion de la Tranche Ferme

5 Contacts

Introduction
Evolutions de OCaml recommandées par LaFoSec

L'étude LaFoSec propose 30 recommandations d'évolution pour le langage OCaml ou ses outils associés. On peut les classer en trois catégories :

- évolutions faciles à implémenter
  - 5 sans impact sur les programmes normaux
  - 5 avec impact : à activer explicitement
- 8 évolutions à long terme
- 12 évolutions à étudier

Introduction

Evolutions

Conclusion

Conclusion de la Tranche Ferme

Contacts

1 Introduction

2 Evolutions

3 Conclusion

4 Conclusion de la Tranche Ferme

5 Contacts

Evolutions
Evolutions faciles 1

Evolutions sans impact sur les programmes normaux

- avertissement sur les variables globales non utilisées
  déjà fait par A. Frisch, warnings 32 à 37
- avertissement sur les annotations de type
  quand le type déclaré par l'utilisateur est moins général que le type inféré par le compilateur
- ajouter une fonction `Bigarray.mlock`
- avertissement lorsqu'un `open` masque des identificateurs
- borne optionnelle sur le temps de calculs des fonctions de la bibliothèque `Str`

Evolutions
Evolutions faciles 2

Evolutions affectant les performances

- GC : effacement des données déplacées
  différence entre *déplacer* et *recopier* : irrelevante en fonctionnel, très importante en sécurité
- environnement (`CAML_DEBUG_SOCKET`, `OCAMLRUNPARAM`)
- `String.create` et `unsafe_*` (pas si simple)
- encapsulation obligatoire de chaque `.ml` par un `.mli`
- ajout d'une fonction `dprintf` pour le debug

Evolutions

Evolutions à long terme 1

- analyse statique des exceptions
  - analyse des exceptions levées par chaque fonction
  - analyse des types des données transportées par les exceptions (fuite de données)
- vérification des débordements d'entiers
- sérialisation/désérialisation typées

  pour éviter les erreurs, et les attaques si on s'assure que la donnée ne peut pas être modifiée après sérialisation
- désérialisation blindée

  pour résister aux entrées non contrôlées

Evolutions
Evolutions à long terme 2

- suppression des initialisations/codes inutilisés dans le runtime (ex : flottants)
  - pas facile à faire : touche toute la chaîne de compilation
  - utile au-delà de la sécurité : codes embarqués, processeurs sans flottants, programmes stand-alone
- annotation explicite de la récursion terminale
- documenter `camlp4`, l'utiliser pour instrumenter les recommandations
- vérificateur de byte-code OCaml

Evolutions

Evolutions à étudier

- chaînes de caractères non mutables
- renforcement de l'encapsulation
  suppression des comparaisons polymorphes
- définition d'un noyau du langage pour la sécurité
  pour éviter les constructions douteuses ou difficiles à évaluer
- développement d'un outil d'analyse statique dédié à la sécurité
- byte-code typé
- vérification du code chargé dynamiquement
- cloisonnement de la mémoire
- contrôle du code d'initialisation des modules liés au programme
- typage avec unités de mesure
- outils de test, debug, profilage

Introduction

Evolutions

Conclusion

Conclusion de la Tranche Ferme

Contacts

1 Introduction

2 Evolutions

3 Conclusion

4 Conclusion de la Tranche Ferme

5 Contacts

Conclusion
Evolutions de OCaml

- certaines évolutions sont déjà en cours
- d'autres sont faisables (question de moyens)
- pour aller plus loin, deux stratégies possibles :
  - définir un sous-ensemble de OCaml et développer les outils d'analyse correspondants
  - un nouveau langage basé sur OCaml

Introduction

Evolutions

Conclusion

Conclusion de la Tranche Ferme

Contacts

1 Introduction

2 Evolutions

3 Conclusion

4 Conclusion de la Tranche Ferme

5 Contacts

Conclusion de la Tranche Ferme

- une liste de problèmes “déjà connus” mais jamais rassemblés dans un même document
  exemple : une optimisation de la compilation des modules permet de casser l'encapsulation : problème connu au niveau de l'efficacité, mais pas comme problème de sécurité
- l'étude porte sur une partie relativement restreinte du langage, pas de :
  - foncteurs
  - variants polymorphes
  - arguments étiquetés
  - GADT
  - ...
- importance de la relecture du code pour la certification

LaFoSec : un point de départ

- OCaml : des apports indéniables pour la sécurité des applications
- des recommandations d'utilisation
- des propositions d'évolution pour OCaml
- une idée à creuser : SecureML
  - quels traits faudrait-il retirer de OCaml ?
  - quels traits faudrait-il ajouter ?
  - quelles garanties demander au compilateur ?
  - quels outils auxiliaires implémenter ?
  - comment le rendre utilisable par l'industrie ?

Introduction

Evolutions

Conclusion

Conclusion de la Tranche Ferme

Contacts

1 Introduction

2 Evolutions

3 Conclusion

4 Conclusion de la Tranche Ferme

5 Contacts

Contacts

- Véronique Delebarre : veronique.delebarre@safe-river.com
- Christèle Faure : christele.faure@safe-river.com