Typage et Analyse Statique

Cours 1

Emmanuel Chailloux

Spécialité Science et Technologie du Logiciel
Master mention Informatique
Université Pierre et Marie Curie

année 2017-2018

Informations sur le cours TAS

Sites

`https://www-master.ufr-info-p6.jussieu.fr/2017/tas`

`www-apr.lip6.fr/~chaillou/Public/enseignement/2017-2018/tas`

Equipe pédagogique

- cours : **Emmanuel Chailloux** (LIP6), **Antoine Miné** (LIP6)
- TME : **Emmanuel Chailloux** (LIP6), **Romain Demangeon** (LIP6), **Sarah Zennou** (Airbus Group)

Description du cours TAS (1)

- Fiabilité logicielle : garantir la sûreté d'exécution (et la sécurité)
  - par tests (analyse dynamique)
  - par preuve totale à la main ou avec des systèmes d'aide à la preuve (Coq, Isabelle, Atelier-B, PVS, ...)
  - par preuve partielle
    - model-checking
    - interprétation abstraite

E.W. Dijkstra : « Program testing can be used to show the presence of bugs, but never to show their absence !»

Théorème de Rice : On ne peut pas définir un algorithme qui pour tout programme P et toute propriété H, renvoie vrai si P vérifie H et faux sinon.

Description du cours TAS (2)

- Génie Logiciel :
  - pour combiner l'approche "programmation modulaire et composition" avec la compilation séparée : vérification des types
  - avec le maximum de généricité : classes de polymorphisme
- Fiabilité du logiciel
  - le typage est une analyse statique
  - la plupart des analyses ont besoin de travailler sur un programme typé
- Efficacité
  - le typage peut permettre des optimisations
  - certaines propriétés statiques aussi

Description du cours TAS (3)

- objets d'étude : les programmes
- vérification :
  - Typage des composants au sens large (Génie Logiciel, Fiabilité des logiciels)
    - fonctions, objets, modules
    - système de types, inférence
    - classes de polymorphisme
  - Analyse statique (Fiabilité des logiciels)
    - par interprétation abstraite (abstraction des valeurs),
  - effectuée automatiquement par d'autres programmes
  - et statiquement : c'est-à-dire avant l'exécution

Description de la partie Typage (1)

- Apprécier la sûreté et la réutilisation apportée par le typage statique
- Comprendre les différentes classes de polymorphisme : paramétrique, objet, ad hoc, de rangées, borné, et leurs utilisations pour la conception de bibliothèques réutilisables
- Savoir lire la définition formelle d'un système de types et pouvoir l'étendre
- Savoir écrire un typeur à partir de sa spécification

mots clés :

systèmes de types, vérification de types, inférence de types, classes de polymorphisme : paramétrique, ad hoc, objet, modèles de programmation : fonctionnel, impératif et objet, interopérabilité des langages

Description de la partie Typage (2)

Quelques exemples récents de langages (+ ou -) typés :

- Hack (Facebook)
- Flow (Facebook), TypeScript (Microsoft)
- Dart (Google), Go (Google), Kotlin (Android)
- Rust (Mozilla)
- Swift (Apple)
- Java / C# (Oracle / Microsoft)
- Scala (EPFL)
- OCaml (Inria/Cambridge), F# (Microsoft), Haskell (UK, Ecosse), Ada (AdaCore), ...

Description de la partie Analyse Statique (1)

L'objectif de ce cours est de présenter les techniques de base de l'interprétation abstraite ainsi que ses fondements théoriques.

L'accent sera mis sur la mise au point pratique d'un analyseur statique plutôt que sur les résultats théoriques nécessaires.
Points abordés :

- sémantique des langages de programmation.
- domaines abstraits relationnels ou non.
- itérateurs de point fixe et accélération de la convergence.
- combinaison d'analyses.

Description de la partie Analyse Statique (2)

- bug d'Ariane 5 (4.6.96) :
  - 37s de vol puis déviation et explosion
  - mauvaise conversion d'un nombre flottant vers un entier non signé 32 bits (l'ordinateur de vol a considéré que l'altitude était négative)
  - reprise du code d'Ariane 4 (entiers sur 16 bits)
- autres bugs fameux : missile Patriot (eereur sur le calcul du temps en flottant), USS Yorktown (div par zéro)
- des environnements :
  - Frama-C (CEA), Newspeak (Airbus) , TrusInSoft Analyser) (TIS), Astrée (Absint),

Exemple : fact

- pas besoin de l'état exact du prog pour prouver une propriété
- on peut perdre de l'information de manière conservative

```
1  int fact(int n) {                  // n!=0
2    int r,i;
3                                       // n > 0
4    r = 2;i=1;
5                                       // n>0, r>0, i>0
6    for (i=3; i<=n; i++) {
7                                       // n>0, r>0, i>0
8      r=r*i;
9                                      // n>0, r>0, i>0
10 }
11 return r;
12     // r != 0  <-- propri\'et\'e \`a prouver
13 }
```

fact(34) == 0

- choisir l'information à conserver et les propriétés à prouver
- sans oublier des états du prog (fact(34) == 0 sur entiers 32 bits)

Plan du cours (1)

1. λ-calcul et codage des donnéees
2. du λ-calcul aux langages de programmation
   typage d'un mini-ML pur (polymorphisme paramétrique)
3. Extension aux traits impératifs
4. sous-typage structurel (polymorphisme de rangées en OCaml) et sous-typae nominal (à la Java) en objet
5. Résolution de la surcharge en Java (et Haskell), génériques en java (polymorphisme paramétrique et borné)
6. interopérabilité (et fusion) des modèles (interopérabilité, Scala), typage progressif, présentations de nouveaux systèmes de types
7. présentation des réalisations et des articles

Plan du cours (2)

8 & 9. Introduction - Sémantique concrète - Abstraction de sémantique

10. Analyse de boucles (accélération de convergence, analyse d'intervalles)
11. Combinaison de domaines abstraits - domaines disjonctifs
12. Domaines relationels (numériques, ...)
13. Analyse de pointeurs, de tableaux
14. Application : vérification de logiciels embarqués avioniques

- réalisation d'un analyseur en TME (à partir du premier TME)
- présentation d'un article
- interrogation écrite courte (en début de séance 12)

Logiciels du cours

Logiciels

- langages d'implantation : OCaml ou F# (ou Swift ?) ou Scala
- langages d'étude : OCaml, Haskell, Java, Scala, C
- environnements : Emacs ou Eclipse (ou VB)
- applications Web : évaluateur λ-calcul, ...
- analyseur maison : noyau fourni (en OCaml)

Evaluation

1. Typage (50%)
   - Examen écrit : 30%
     2h sur papier - documents autorisés
   - réalisation, prise en main d'environnement ou synthèse d'article : 20%
     rapport + soutenance
2. Analyse statique (50 %)
   - contrôle continu : interrogation(s) : 10%
   - présentation d'article : 20%
   - projet initié en TME : 20%

Des références (1)

- vidéos
  - vidéos de Gérard Berry au Collège France, `http://www.college-de-france.fr/site/gerard-berry/index.htm` en particulier pour ce cours l'années 2009-2010)
- transparents
  - 1/2 journée industrielle sur "Méthodes formelles et langages pour le développement de logiciels fiables dans l'industrie" :

`https://ihp2014.pps.univ-paris-diderot.fr/doku.php?id=industry` `dizaine d'intervenants industriels`

- ouvrages de référence :
  - Types and Programming Languages
    B. Pierce
    The MIT Press.
  - Principles of Program Analysis
    F. Nielson, H. Nielson et C. Hankin
    Springer.

Des références (2)

- articles fondateurs :
  - A Theory of Type Polymorphism in Programming
    Robin Milner
    J. Comput. Syst. Sci. 17(3) : 348-375 (1978)
  - Abstract interpretation : a unified lattice model for static analysis of programs by construction or approximation of fixpoints.
    Patrick Cousot and Radhia Cousot, POPL, 1977.

- événements proches :
  - conférence Patrick Cousot jeudi 29/9/16 à 18h amphi 15.
  - MOOC OCAML à partir du 26/9/2016 :
    `http://tinyurl.com/ocamlmooc2` (VOSTF)
  - conférence POPL 2017 (du 15 au 21 janvier) à Jussieu
    `http://conf.researchr.org/home/POPL-2017`
    (appel à étudiants volontaires)