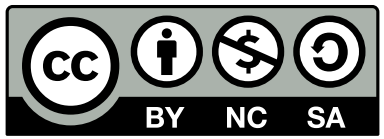
Chapitre 5 - Analyse sémantique et typage

INF7641 Compilation

Jean Privat

Université du Québec à Montréal

INF7641 Compilation
v251



Plan

Analyse sémantique

Analyse sémantique

- Extraire et valider du « sens » d'un programme

Objectifs

- Conformité avec la spécification du langage
  - Lexer et parser ne suffisent pas forcément
  - Idéalement avec des messages d'erreurs utiles
- Pré-traitement du code avant l'exécution
  - Pour plus de performance

Exemples

- Évaluation et validation des littéraux
- Utilisation de variables déclarées
- Portée (et masquage) des variables
- Variables initialisés avant utilisation
- Conformité des opérations et de valeurs (typage statique)
- Présences de `return` dans tous le chemins
- Et tout ce qui peut simplifier la vie de l'interpréteur ou du compilateur

Mise en œuvre

- Étapes préalable à l'interprétation (et la compilation), sous formes de passes
- Passes idéalement indépendantes pour meilleure ingénierie logiciel (couplage, cohésion, modularité)
- Implémentation de visiteurs concrets dédiés

Bug Pharo

If you create a new Pharo 10 image, open a Playground paste the following commit SHA, it will freeze the image: 49781e985875315d06c5991f094c30d6f8c9f02d

— Pharo#12064, 2022

Traitement des littéraux

Étendre MiniC

- Valider que les littéraux sont sains avant l'exécution
  - Afficher un message d'erreur sinon
- Précalculer la valeur des littéraux
  - Ne pas `parseInt` à chaque fois dans `caseInt`
  - Imaginez le gain dans une boucle!

Mise en œuvre

- Un visiteur `LitteralAnalysis`
- Traiter seulement les `caseInt`
- Stocker les valeurs des littéraux quelque part (`HashMap`)
- Modifier l'interpréteur pour profiter de la valeur déjà calculée

Portée de variables (scope)

Étendre MiniC

- Pour être utilisable, une variable doit être déclarée avant
- La portée d'une variable va de la déclaration à la fin du bloc
- Le masquage (*shadowing*) est autorisé

Mise en œuvre

- Un visiteur `ScopeAnalysis`
- Une gestion des portées courantes
  - Classe `Scope`, simplement chaînée
  - Pour connaître les variables valides
  - Et les mettre à jour (nouvelle variable, fin de bloc)
- Quoi d'autre ?

Mise en œuvre

- Un visiteur `ScopeAnalysis`
- Une gestion des portées courantes
  - Classe `Scope`, simplement chaînée
  - Pour connaître les variables valides
  - Et les mettre à jour (nouvelle variable, fin de bloc)
- Quoi d'autre ?
  - Distinguer les variables de leurs noms: une classe `Variable`
  - Modifier l'interpréteur

Typage statique

Étendre MiniC

- Ajouter un type `bool` (en plus du type `int`)
- Ajouter deux littéraux `true` et `false` de type `bool`
- Ajouter la primitive `printbool()` (qui prend un `bool`)
- Les opérations arithmétiques prennent et retournent un `int`
- Les < prennent deux `int` et retournent un `bool`
- Les condition des `if` et `while` doivent prendre un `bool`
- Les expressions affectées doivent respecter le type des variable

Mise en œuvre

- Un visiteur `TypeAnalysis`
- Associer chaque expression et chaque variable à un type statique
- Valider les règles de typage
- Quoi d'autre ?

Mise en œuvre

- Un visiteur `TypeAnalysis`
- Associer chaque expression et chaque variable à un type statique
- Valider les règles de typage
- Quoi d'autre ?
  - Implémenter la primitive `printbool` de l'interpréteur
  - Et les expressions `true` et `false`
  - Pas d'autres modification nécessaires

Définition et appel de fonctions

Étendre MiniC

- Généraliser la définition de fonctions
  - Toujours un type de retour, pour simplifier
- Ajouter l'appel de fonction : expression et instruction
- Ajouter `return` avec une valeur
- Note : si une fin de fonction est atteinte sans `return`, alors `0` ou `false` est retourné

Mise en œuvre

- Créer une classe `Function`
- Étendre `ScopeAnalysis`
  - Collecter les définitions de fonctions `Map<String, Function>`
  - Collecter les paramètres dans un `Scope`
- Étendre `TypeAnalysis`
  - Valider les arguments des appels
  - Valider le type du retour
- Étendre l'interpréteur
  - Appels : transfert de contrôle et passage d'arguments
  - Retours
  - Empiler des cadres de pile (*frame*)