CSC4251_4252 : Analyse Sémantique

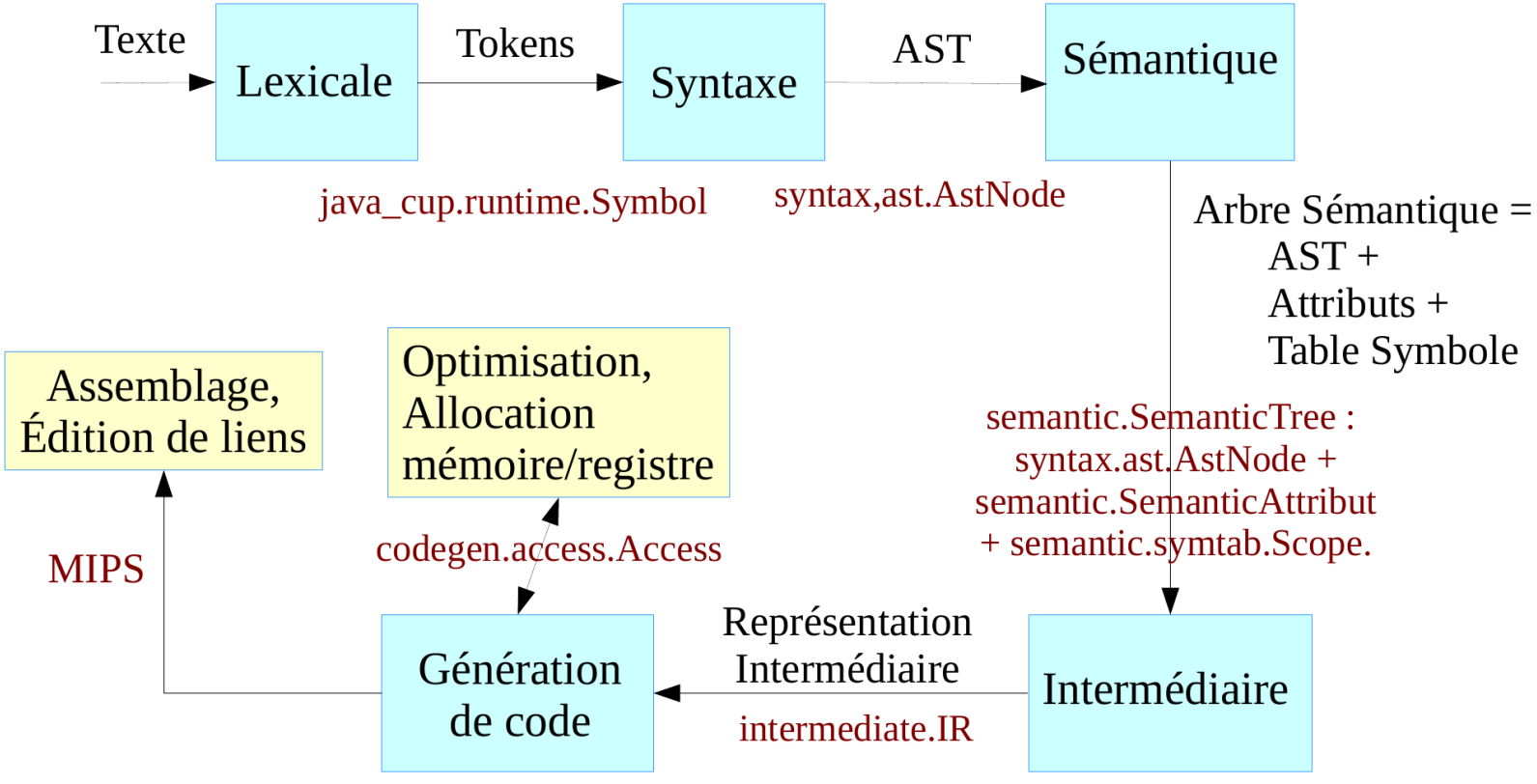
Pascal Hennequin, J. Paul Gibson, Denis Conan

Novembre 2024

Sommaire

1 Où en est-on ? I

- **Retour : Phases de Compilation**



1 Où en est-on ? II

- **Compilation Classique versus Moderne**
  - « *Dragon Book* », Aho, Ullman, 1977
  - « *Modern Compiler* », Appel 2000, ...
- **Identique pour l'analyse lexicale et syntaxique**
- **Différences importantes sur l'analyse sémantique et la génération de la forme intermédiaire**
- **Différences sur les solutions techniques et aussi sur le problème à résoudre**

1 Où en est-on ? III

- **Compilation Multi-passe ou Mono-passe ?**
  - Exécution en séquence *vs.* chaînage (pipeline) des phases
  - Mono-passe : contraignant sur l'algorithmique du compilateur, mais économe sur les performances en temps, en espace, en entrée-sortie
  - Multi-passe : plus souple dans la programmation et l'indépendance des phases de compilation
- **Historiquement : Mono-passe était un enjeu vital**
  - Mais, contrainte sur le langage : Tout ce qui est nécessaire pour analyser une portion de programme doit exister « avant » dans le source
  - Mais, algorithmique lourde pour tout résoudre en une fois
- **Aujourd'hui : Multi-passe encouragé**
  - Contraintes plus faibles sur les performances
  - Donc, langages plus « libéraux » (p.ex. JAVA)

2 Analyse sémantique I

- **Objectifs :**
  - Valider les règles du langage non gérables au niveau syntaxique
  - Établir la signification du programme et sa sémantique d'exécution
  - Construire les éléments de contexte
    - Informations nécessaires pour l'exécution mais non explicites dans l'arbre de syntaxe, ou explicites dans une autre partie de l'arbre
  - Vérifier ou Inférer des propriétés de l'algorithme
    - Terminaison, validité de l'algorithme, non débordement, ...
    - N.B. : Théorème de Rice $\Longrightarrow$ problèmes indécidables

2 Analyse sémantique II

- **De la théorie :**
  - « Haute » : lambda-calcul, théorèmes du point fixe, preuve de programmes
  - « Basse » : grammaires attribuées, traduction dirigée par la syntaxe
- **Mais en pratique ?**
  - Trop théorique, et peu de résultats pour de « vrais » programmes
  - Trop orientée sur la contrainte « compilation mono-passe »
- **Conclusion : approche pragmatique**
  - Des fonctions sémantiques réalisées de façons modulaires et en passes successives

3 Attributs sémantiques

- **Analyse Sémantique = Décoration de l'AST**
  - Calcul d'attributs associés aux différents nœuds de l'arbre de syntaxe
  - Visibilité des identificateurs
  - Type de données
  - Chemins d'exécution
  - ...
- **Attribut Synthétisé *vs.* Attribut Hérité**
  - Hérité : la valeur est construite à partir de la valeur du père
  - Synthétisé : la valeur est construite à partir des valeurs des fils
  - Mixte ? Essayer de décomposer en (Hérité + Synthétisé)
  - N.B. : Analyse syntaxique LR $\Longrightarrow$ synthétisé facile, hérité difficile
- **Algorithmie : Parcours récursif en profondeur d'Arbre**
  - Cf. Mémento du projet Minijava pour la réalisation

4 Analyse statique ou dynamique

- **Statique**
  - Les attributs sémantiques sont calculés sur la structure statique de l'AST
- **Dynamique**
  - On ajoute à l'AST, les chemins d'exécutions du programme pour calculer des attributs qui vont hériter de nœuds traversés « avant » ou « après » à l'exécution
  - Algorithmique :
    - « Couture d'arbre »
    - Interprétation symbolique
    - Équations de flots de données
    - Analyse de vivacité

5 Des fonctions sémantiques I

Liaison des identificateurs

- Table des symboles
- Détections : « non défini », « déjà défini », « initialisation »
- Consistance des définitions : détection de boucle (héritage JAVA, définitions régulières JFlex, etc.), ...
- Paradigmes orientés objet : héritage (simple *vs.* multiple), redéfinition (*override*), surcharge

Contrôle de Type

- Typage des expressions, contraintes d'affectation et d'appel
- Surcharge des opérateurs
- Transtypage (*cast*) implicite, équivalence des types
- Héritage Objet

5 Des fonctions sémantiques II

- **Propagation de constantes**
  - Évaluation « immediate » de l'assembleur, optimisation d'expressions
  - Calcul d'intervalles : réduction des contrôles à l'exécution, optimisation de boucle, ...
  - Élimination de code
- **Analyse de vie, Analyse de dernière définition**
  - Optimisation mémoire : registre / sauvegarde de registre / pile / tas
  - Variable *unused*, p.ex. paramètre d'appel
  - *Garbage Collect*
- **Optimisation appels de fonctions**
  - Fonction *vs.* recopie/insertion de code
  - Détection / Optimisation de récursivité

5 Des fonctions sémantiques III

- **Vivacité, Analyse de flot de données/contrôle**
  - Code mort
  - Existence `return` dans toutes les exécutions d'une méthode
  - ...
- **Signalement de sémantiques douteuses : Compromis entre optimisation silencieuse et détection d'erreurs de programmation**
  - Boucles infinies, code mort, ...
  - Transtypages « étranges »
  - Expressions à valeurs triviales

6 Table des symboles

- **Liaison entre déclarations et utilisation des identificateurs**
  - Et mise en œuvre de la surcharge, de la redéfinition, ...
- **Attribut « mixte »**
  - Une déclaration est synthétisée pour trouver sa portée (*scope*) – La visibilité est héritée, et éventuellement synthétisée dans l'AST

$\Longrightarrow$ **Structure spécifique : « Table des symboles »**

  - Regroupe l'ensemble des déclarations d'identificateurs, leurs portées et leurs visibilités depuis chaque nœud de l'arbre
  - Propagée dans la suite de la compilation (y compris l'exécutable)
- **Dans le cas d'un langage orienté objet**
  - La visibilité est aussi héritée en suivant l'arbre d'héritage des classes qui est indépendant de l'AST (information de contexte)
  - Et aussi liaison dynamique, ...

7 Contrôle de type I

- **Langage compilé $\Longleftrightarrow$ Langage fortement typé**
- **Typage**
  - Partie importante en volume de la définition d'un langage
  - Mais beaucoup de règles rapides à vérifier, bien que difficilement gérables au niveau syntaxique
- **Valider les contraintes de typage du langage**
  - Conformité des opérateurs
  - Affectations
  - Appel de fonctions
- **Fixer les contraintes pour l'implantation des variables :**
  - Type $\Longrightarrow$ taille mémoire, type de registre, ...
  - Construction d'un attribut synthétisé

7 Contrôle de type II

- **Valider et réaliser les transtypages implicites ou explicites :**
  - Équivalence de types, héritage
  - Fonctions de conversion (transtypage ou *cast*, *autoboxing*)
- **Calcul de la sémantique des expressions**
  - Surcharge des opérateurs
- **Gérer les mécanismes de construction de type du langage**
  - Références, tableaux, enregistrements/structures, énumérations, ensembles, ...
  - Héritage multiple