Langages : interprétation et compilation

Pablo Rauzy
pr@up8.edu
pablo.rauzy.name/teaching/liec

UNIVERSITÉ PARIS 8 VINCENNES-SAINT-DENIS
UFR MITSIC / L3 informatique

Séance 7
Écriture d'un interpréteur

Écriture d'un interpréteur

Représentation interne du code

- Une fois l'analyse sémantique effectuée, on obtient notre représentation interne du code : l'*arbre de syntaxe abstraite*.
- Cette représentation interne s'appellerait représentation *intermédiaire* dans le cas d'un compilateur.
- C'est souvent sur cette représentation là qu'on applique des transformations du code telles que des optimisations.

Interprétation

- Dans le cas d'un interpréteur, on peut souvent directement interpréter l'AST dans notre langage hôte.
- Cette interprétation est d'autant plus facile que les précédentes analyses ont relevées les erreurs possibles.
  - Par exemple, plus besoin de vérifier si une variable existe bien quand elle est utilisée si cette vérification a déjà eu lieu lors de l'analyse sémantique.

- Comme pour l'analyse sémantique, la méthode va être de *traverser* l'AST du code à interpréter.
- Cette fois-ci, au lieu de faire des vérification, on va utiliser le langage *hôte* pour évaluer les expressions et effectuer les instructions de l'AST.
- On va donc aussi avoir besoin d'un environnement, mais cette fois il servira à stocker des *valeurs* et plus seulement des informations de typage.

- À votre avis, à quoi un interpréteur peut servir dans le cadre d'un compilateur ?

Interprétation et compilation

- À votre avis, à quoi un interpréteur peut servir dans le cadre d'un compilateur ?
  - Faire des optimisations grâce à *évaluation partielle*.

Évaluation partielle

- On peut voir un programme comme une fonction qui prend deux types d'entrées :
  - les entrées *statiques* (connues à la compilation), et
  - les entrées *dynamiques* (connues seulement à l'exécution).
- Notons $P : E_{\text{statique}} \times E_{\text{dynamique}} \to S$.
- L'idée de l'évaluation partielle est de *spécialiser* cette fonction P en une fonction $P\star : E_{\text{dynamique}} \to S$ en précalculant tout ce qui peut l'être lors la compilation.
- Le programme résultant $P\star$ est plus rapide puisqu'il a moins de calcul à faire.

Projections de Futamura

- Yoshihiko Futamura a publié en 1971 un article amusant où il utilise l'évaluation partielle et présente 4 *projections*.
- Son idée est de prendre comme programme... un interpréteur.

Projections de Futamura

- Yoshihiko Futamura a publié en 1971 un article amusant où il utilise l'évaluation partielle et présente 4 *projections*.
- Son idée est de prendre comme programme… un interpréteur.
- Si P est un interpréteur, on peut prendre comme E_{statique} le code source qu'il doit interprété et ses entrées statiques, et comme $E_{\text{dynamique}}$ les entrées dynamiques du programme à interpréter.
- À quoi correspond $P\star$ dans ce cas ?

Projections de Futamura

- Yoshihiko Futamura a publié en 1971 un article amusant où il utilise l'évaluation partielle et présente 4 *projections*.
- Son idée est de prendre comme programme… un interpréteur.
- Si P est un interpréteur, on peut prendre comme E_{statique} le code source qu'il doit interprété et ses entrées statiques, et comme $E_{\text{dynamique}}$ les entrées dynamiques du programme à interpréter.
- À quoi correspond $P\star$ dans ce cas ?
  - Une version "compilée" de $\langle P, E_{\text{statique}} \rangle$!
  - C'est à dire une version de E_{statique} écrite directement dans le langage de P et plus rapide que quand P l'interprète.
  - Cela correspond à la première projection de Futamura.
- Avez-vous une idée de ce que peuvent être les autres ?

Les projections de Futamura

1. Spécialiser un interpréteur pour un code source particulier donne un exécutable.
2. Spécialiser un spécialiseur pour un interpréteur donne un compilateur.
3. Spécialiser un spécialiseur pour lui même donne un “compilateur de compilateur” (qui prend n'importe quel interpréteur et en fait un compilateur).
4. Spécialiser ce dernier outil sur lui même est une opération idempotente donc on obtient une sorte de *Quine*.

Codons !

→ Écrivons ensemble un interpréteur simple pour notre langage vu lors de la séance 5.