Théorie des langages de programmation
Le projet (III)

Alain Chillès – 祁冲

ParisTech Shanghai Jiao Tong
上海交大–巴黎高科卓越工程师学院

9 novembre 2020 – 2020年11月9日 – 庚子九月廿四

Plan

Python Java C Prolog Forth C++ Ada Pascal APL Lisp Fortran

Projet

Que faut-il faire pour interpréter ?

Interprète – mode simple

- Interaction avec l'environnement, on connaît tout (table des symboles, piles, machine virtuelle, etc.)
- pas de définition de nouvelles fonctions, uniquement les fonctions de base ;
- il faut séparer les informations !

Exemple

La fonction $+$

- Prend deux entiers sur la pile de données, les additionne et met le résultat sur la pile de données.

Répartition des informations

- Dans la table des symboles, on doit mettre :
  - Le nom;
  - l'adresse où est le code ;
- Dans la machine virtuelle, on doit mettre
  - Le code ;
  - et... ?

Objets de base : seront définis en langage C

1. Les objets qui seront sur la pile de données : les entiers, les adresses des chaînes de caractères (où seront rangées les chaînes de caractères ?) ;
2. les fonctions de base : dup, drop, swap, ., count, type, $=$, $*$, $+$, $-$, etc.
3. les structures de contrôles seront réservées à la compilation...

Problème des chaînes de caractères

Le rangement usuel en `Forth`(et donc en L_{AC}) est la longueur, suivie des codes ascii des caractères la composant.

Exemple

La chaîne `"Alain"` sera rangée sous la forme :

5	65	108	97	105	110

En langage `C`, elle sera rangée sous la forme :

65	108	97	105	110	0

Mots de base

- dup (n – n n) : duplique le sommet de la pile de données.
- drop (n –) : supprime le sommet de la pile de données.
- swap (n1 n2 – n2 n1): échange les deux éléments sur le haut de la pile de données.
- . (e –) : imprime l'entier qui est sur le sommet de la pile de données et le supprime.
- count (ad – ad' e) : prend l'adresse de chaîne en haut de la pile, et la remplace par l'adresse du début de la chaîne sans la longueur et met la longueur sur le sommet.
- type (ad e –) : imprime *e* caractères dont les codes sont à partir de *ad*.
- = (e1 e2 – b) : teste si $e_1 = e_2$ et empile un booléen.

Choix simples

- Sur la pile des données, il n'y aura que des entiers (qui pourront être lus comme des entiers, des adresses de chaînes, des booléens, etc.)
- Sur la pile de retour, il n'y aura que des entiers.
- Dans la table des symboles et la machine virtuelle, il n'y aura que des entiers (pour simplifier la lecture et l'écriture).

Exemple de rangement : la fonction +

1. Dans la table des symboles

Numéro	0	1	2	3
LAC	0	1	43	0
Signification		l	+	c

Où $l = 1$ est la longueur du nom, 43 est le code de + et c l'endroit où trouver dans la machine virtuelle le code de +. (c se nomme le Cfa de + – Code Field Address).

2. Dans la machine virtuelle

Numéro	0	1
VM	0	0

Où le premier 0 signale que c'est une fonction de base (codée en C) et le deuxième 0 son numéro parmi les fonctions de base.

Exemple de rangement : la fonction `swap`

1. Dans la table des symboles

Numéro	4	5	6	7	8	9	10
LAC	1	4	115	119	97	112	2
Signification	*a*	*l*	s	w	a	p	*c*

Où $a = 1$ désigne l'adresse dans la table des symboles où commence le nom de la fonction précédemment définie (ici $+$), et $c = 2$ désigne le Cfa de `swap`.

2. Dans la machine virtuelle

Numéro	0	1	2	3
VM	0	0	0	1

Interprète sans définition de fonctions

On veut interpréter ma phrase suivante :

2␣3␣4␣+␣*␣.

Comment faire ?

1. Analyse lexicale : on obtient 6 lexèmes :

   "2" "3" "4" "+" "*" "."

2. Analyse syntaxique : il n'y en a pas !
3. Analyse sémantique (se fait ici en même temps que l'exécution, lexème par lexème).
   - "2" existe et correspond au nombre 2 (de même, 3 et 4), 2 est empilé (de même 3 et 4) ;
   - "+" existe et correspond à la fonction 0 du processeur, on exécute la fonction 0 du processeur, etc.

Installation des fonctions de base

On peut concevoir une fonction en langage C qui permet de remplir correctement la table des symboles LAC et la machine virtuelle VM. Notons la declare_base

Source C

```c
void declare_base(char nom []);
```

La fonction + peut alors être déclarée par :

Source C

```c
declare_base("+");
```

Comment définir une fonction du processeur ?

Source C

```c
// Définition du processeur
typedef void (*base)(void); // Type de la
↪ fonction de base
base processeur[50]; // Taille à préciser
```

Source C

```c
// Supposons la fonction plus bien définie
// On la place dans le processeur
processeur[0] = &plus;
```

Source C

```c
// Utilisation de la fonction du processeur
processeur[0]();
```

Comment trouver si une fonction est déjà définie ?

Si la table des symboles est définie par :

Numero	0	①	2	3	4	5	6	7	8	9	10
LAC	0	1	43	0	①	4	115	119	97	112	2
Signification		l	+	c	a	l	s	w	a	p	c

Cherchons la fonction +.

- On commence à la dernière adresse définie (ici 5, début de la fonction `swap`) ;
- on cherche si c'est + (1 | 43)... non ? on prend la valeur de la case $4 = 5 - 1$, c'est 1 (début de la fonction +), est-ce + ? Oui ! Son Nfa vaut donc 1, et on calcule son Cfa qui vaut 0 – case 3, obtenu en comptant à partir du Nfa, en tenant compte de la longueur du nom.

Cherchons la fonction 1. Le même procédé, nous fait passer par l'adresse 5, puis 1. On arrive en 0 (LAC[0]). La fonction n'est donc pas définie !

Comment exécuter ?

1. "2" n'est pas une fonction, c'est un entier : on l'empile sur la pile de données (de même avec 3 et 4) ;
2. "+" est une fonction ;
3. on trouve le Cfa de +, c'est 0. On va à l'adresse VM[0], on y trouve 0, c'est donc une fonction de base, son numéro est en VM[1], c'est 0. On exécute processeur[0] ;
4. "*" n'est pas définie, on génère une erreur et on vide les piles de données et de retour...
5. on définit $*$ et ., et on recommence... Cela doit marcher !

Exécution de L_{AC}

L_{AC} – Exemple d'exécution

```
1  2 3 4 + * . -> 14  ok
2  2 3 4 * - 5 6 + * . -> -110  ok
```