Typage, table des symboles et analyse sémantique

Alexis Nasr
Carlos Ramisch
Manon Scholivet
Franck Dary

Compilation – L3 Informatique
Département Informatique et Interactions
Aix Marseille Université

Analyse sémantique

Phase de la compilation qui consiste à vérifier que les constructions d'un programme sont sémantiquement cohérentes

- Un programme peut être syntaxiquement correct mais contenir des erreurs "sémantiques"
  - Variables/fonctions non déclarées
  - Fonctions avec mauvais nombre/type de paramètres
  - Opérateurs appliqués à des types incompatibles (`int + string`)
  - Pas de `return` alors que la fonction n'est pas `void` (ou vice-versa)
  - Pas de `main`
  - ...

Analyse sémantique

Il n'est pas possible ou pratique de vérifier certaines contraintes avec une grammaire hors contexte

- Comment vérifier, par exemple, dans la grammaire que "toute variable utilisée doit être déclarée" ?

Sous-tâches

- Remplissage et consultation de la table des symboles
  - Toute variable/fonction utilisée est déclarée
  - Toute fonction est appelée avec le bon nombre de paramètres
  - Calcul des adresses des variables locales/globales
- Application des règles du système de typage
  - Compatibilité entre types déclarés et utilisations des variables
  - Compatibilité entre opérateurs et opérandes dans expressions
  - Compatibilité entre paramètres réels et déclarés des fonctions
  - Compatibilité entre valeur retournée et type de la fonction déclarée
  - Conversions et coercitions automatiques

Table des Symboles

Alexis Nasr
Carlos Ramisch
Manon Scholivet
Franck Dary

Compilation – L3 Informatique
Département Informatique et Interactions
Aix Marseille Université

La table des symboles

- Elle rassemble toutes les informations utiles concernant les variables et les fonctions du programme.
- Pour toute variable, elle garde l'information de :
  - son nom
  - son type
  - sa portée
  - son adresse en mémoire
  - la taille mémoire qu'elle occupe
- Pour toute fonction, elle garde l'information de :
  - son nom
  - sa portée
  - le nom et le type de ses arguments, ainsi que leur mode de passage
  - le type du résultat qu'elle fournit

Consultation / Ajout

- Elle grandit pendant la compilation des parties déclaratives :
  - déclaration de variables,
  - définition de fonctions.
- Elle est consultée pendant la compilation des parties exécutables :
  - appel de fonction,
  - référence à une variable.
- La table des symboles doit être réalisée avec soin : on estime qu'un compilateur passe la moitié de son temps à la consulter.

Table globale et tables locales

Plusieurs tables de symboles coexistent

- La table globale stocke :
  - les variables globales
  - les fonctions
- Chaque fonction définit une table locale qui stocke :
  - les paramètres
  - les variables locales
  - les fonctions locales (si elles sont autorisées par le langage)

Exemple

```
1 entier u;
2 calcul(entier a, entier b)
3 entier u;
4  { u = a - b;
5    retour u;}
6 main()
7  { u = calcul(4, 5);}
```

- En 1, on vérifie qu'il n'existe pas déjà de variable globale `u`.
- Si c'est bien le cas, on l'ajoute à la table des symboles globale.

Exemple

```
1 entier u;
2 calcul(entier a, entier b)
3 entier u;
4  { u = a - b;
5    retour u;}
6 main()
7  { u = calcul(4, 5);}
```

- En 2, on vérifie qu'il n'existe pas déjà une fonction `calcul`.
- Si c'est bien le cas, on l'ajoute à la table des symboles globale
- On vérifie que la fonction `calcul` ne comporte pas déjà de paramètre `a`
- Si c'est bien le cas, on l'ajoute à la table des symboles locale
- Idem pour `b`.

Exemple

```
1 entier u;
2 calcul(entier a, entier b)
3 entier u;
4  { u = a - b;
5    retour u;}
6 main()
7  { u = calcul(4, 5);}
```

- En 3, on vérifie qu'il n'existe pas déjà de variable locale ni de paramètre u.
- Si c'est bien le cas, on l'ajoute à la table des symboles locale
- S'il existe une variable globale u on peut en avertir l'utilisateur.

Exemple

```
1 entier u;
2 calcul(entier a, entier b)
3 entier u;
4  { u = a - b;
5    retour u;}
6 main()
7  { u = calcul(4, 5);}
```

- En 4, on vérifie qu'il existe des variables locales, des paramètres ou des variables globales ayant pour nom `u`, `a` et `b`.

Exemple

```
1 entier u;
2 calcul(entier a, entier b)
3 entier u;
4  { u = a - b;
5    retour u;}
6 main()
7  { u = calcul(4, 5);}
```

- En 5, on vérifie qu'il existe une variable locale, un paramètre ou une variable globale u.

Exemple

```
1 entier u;
2 calcul(entier a, entier b)
3 entier u;
4  { u = a - b;
5    retour u;}
6 main()
7  { u = calcul(4, 5);}
```

- En 6, on vérifie qu'il n'existe pas déjà une fonction `main`.
- Si c'est bien le cas, on l'ajoute à la table des symboles globale

Exemple

```
1 entier u;
2 calcul(entier a, entier b)
3 entier u;
4  { u = a - b;
5    retour u;}
6 main()
7  { u = calcul(4, 5);}
```

- En 7, on vérifie qu'il existe une fonction `calcul`.
- on vérifie qu'il existe bien deux variables `a` et `b` possédant un type compatible avec la définition de la fonction `calcul`;
- on vérifie qu'il existe bien une variable `u` possédant le même type que la valeur retournée par la fonction `calcul`.

Exemple

```
1 entier u;
2 calcul(entier a, entier b)
3 entier u;
4  { u = a - b;
5    retour u;}
6 main()
7  { u = calcul(4, 5);}
```

- À la fin, on vérifie que la fonction `main` existe

Deux philosophies

- Lors de l'analyse sémantique, on a besoin d'accéder à la table globale et à la table locale courante
- Deux possibilités :
  1. La table locale courante n'existe que pendant le parcours d'une fonction, elle est ensuite détruite.
  2. La table locale de chaque fonction est maintenue à l'issue du traitement de la fonction.
     A l'issue du traitement du programme on a une table globale et autant de tables locales que de fonctions.
- On choisit ici la deuxième option.

Table globale et tables locales

programme source

```
entier T[10],
entier v;



f(entier a, entier b)
entier c, entier k;
{}


main()
entier d;
{}
```

tables des symboles

GLOBALE	T	VAR	10	0
	v	VAR	1	10
	f	FCT	2	
	main	FCT	0	
f	a	PARAM	1	0
	b	PARAM	1	1
	c	VAR	1	0
	k	VAR	1	1
main	d	VAR	1	0

Les adresses des variables

- Comment le compilateur alloue-t-il des adresses aux variables ?
  en comptant l'espace utilisé par les variables rencontrées.
- variables globales

```
entier   x,   t[10],   y,    z;
         0    1        11    12
```

- les arguments de fonctions

```
f(   entier a,   entier b)
     0           1
```

- On ne se soucie pas pour l'instant de l'unité, c'est le nombre d'octets qu'il faut pour stocker un entier

Construction des tables de symboles

- Les tables sont construites lors d'un parcours descendant de l'arbre abstrait.
- On a accès à tout moment à la table globale
- Lorsqu'on parcours le corps d'une fonction, on a accès à une table locale
- Pour chaque référence à une variable, un paramètre ou une fonction on doit vérifier qu'elle (ou il) existe.
- Pour chaque définition de variable, de paramètre ou de fonction, on doit vérifhier qu'elle (ou il) n'existe pas.
- A tout moment, on doit savoir si on se trouve :
  - hors d'une fonction,
  - dans la liste des paramètres d'une fonction,
  - dans le corps d'une fonction.

Typage

Alexis Nasr
Carlos Ramisch
Manon Scholivet
Franck Dary

Compilation – L3 Informatique
Département Informatique et Interactions
Aix Marseille Université

Système de types

- Les *variables*, *fonctions* et *expressions* ont des types associés
- Types : déclarés ou inférés ? modifiables ?
  - Déclarés : identificateurs déclarés avec un type (C, Java)
  - Inférés : types sont inférés selon leur utilisation (Perl, Python, PHP)
  - Modifiables : changement de type dans la même portée (Python)
- Vérifications : statiques (compilation) ou dynamiques (exécution) ?
  - Statiques : compatibilité types utilisés/déclarés des variables
  - Dynamiques : dépassement des bornes d'un tableau
- Conversions de types
  - Implicites (coercitions) : `x = 1 + 5.3`
  - Explicites (cast) : `x = (int)y + 1`
- Polymorphisme (fonctions, opérateurs, types complexes)

Vérifications de types statique

Réalisées pendant la compilation

- Expressions : type(opérandes) = type(opérateur) ?
- Affectation : type (expression) = type (variable) ?
- Appel de fonction : type(arg. réels) = type(arg. formels) ?
- Instructions de contrôle : type(expression) = booléen ?
- On doit pouvoir comparer les types
- Génération de code $\leftrightarrow$ table des symboles

Règles de typage en *L* I

Variables

- Deux types possibles : entier et tableau d'entiers
- Trois portées possibles globale, argument ou locale

1. Lors de la déclaration, vérifier :
   - L'unicité des variables déclarées dans une même portée
     - Il est possible de déclarer une variable locale ou argument qui a le même nom qu'une variable globale
     - Il n'est pas possible de déclarer une variable locale qui a le même nom qu'un argument
     - Uniquement la portée la plus proche est accessible (warning)
   - Un tableau est toujours une variable globale

Règles de typage en L II

Variables

2. Lors de l'appel dans une affectation ou expression, vérifier :
   - Variable utilisée est déclarée (recherche dans l'ordre) :
     1. Variable locale
     2. Argument de fonction
     3. Variable globale
   - Les tableaux ne peuvent jamais être utilisés sans indice
   - Les entiers ne peuvent jamais être indicés
   - Aucune conversion ou coercition possible
   - Dépassement des bornes du tableau - pas de vérification

Règles de typage en *L* III

Fonctions

1. Lors de la déclaration, vérifier :
   - L'unicité des fonctions
     - Pas de *polymorphisme* en *L*
     - Deux fonctions sont identiques si leurs identificateurs sont identiques (indépendamment du nombre de paramètres)

Règles de typage en *L* IV

Fonctions

2. Lors de l'appel dans une instruction ou expression, vérifier :
   - Fonction appelée doit être déclarée *avant* dans le programme
   - Nombre d'arguments réels passés à la fonction appelée est identique au nombre d'arguments formels dans la déclaration
   - Il existe une fonction sans arguments qui s'appelle main
   - Un seul type de retour : entier - pas de vérification
   - Un seul type d'argument : entier - pas de vérification
   - Présence de `retour` et son utilisation dans expression - pas de vérification
   - Toute branche d'exécution contient `retour` - pas de vérification