

Partie VI : Analyse des types

- Règles de typage
- Grammaires attribuées
- Attributs
- Spécification complète de la grammaire contextuelle de misc
- Comment passer d'une spécification à un compilateur

Martin Odersky, LAMP/DI

1

Règles de typage

- La déclaration des identificateurs n'est pas la seule chose à vérifier dans un compilateur.
- En misc, comme dans la plupart des langages de programmation, les expressions ont un type.
- Il faut vérifier que les types sont corrects.
- Exemples:
 - Les opérandes de + doivent être des entiers.
 - Les opérandes de = doivent être compatibles. (`Int`, `Int` est OK, de même que `List[Int]`, `List[Int]`, mais pas `Int`, `List[Int]`)
 - Le nombre d'arguments passé à une fonction doit être égal au nombre de paramètres de cette fonction.
 - etc.
- Comment spécifie-t-on les règles de typage ?

Martin Odersky, LAMP/DI

2

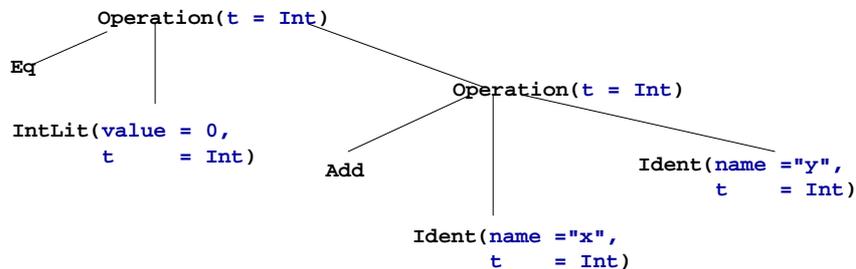
Grammaires attribuées

- On ajoute des attributs et des règles d'attribution aux grammaires non contextuelles.
- A chaque symbole peuvent être attachés des attributs.
- A chaque production peuvent être associées des règles d'attribution qui mettent en relation les attributs des symboles de la production.
- Exemple:

$E(t)$ = Operation Add $E(t_1)$ $E(t_2)$ Operation Eq $E(t_1)$ $E(t_2)$ Ident ident IntLit int	$t_1 = t_2 = \text{Int}, t = \text{Int}$ $t_1 = t_2 = \text{Int}, t = \text{Int}$ $\text{sym} = \text{lookup}(\text{ident}), t = \text{sym.type}$ $t = \text{Int}$
---	---

Grammaires attribuées (2)

- Une *attribution* est une affectation de tous les attributs d'un arbre syntaxique satisfaisant toutes les règles d'attribution.
- Exemple : $0 == (x + y)$ où x et y sont de type `Int`



Grammaires attribuées (3)

- Un programme est légal si :
 - c'est une phrase du langage décrit par la grammaire non contextuelle, et
 - il existe une attribution pour son arbre.
- Un langage est donc caractérisé complètement par sa *grammaire non contextuelle* et sa *grammaire contextuelle*.
- Toutefois, rien n'est encore dit au sujet de la *signification* d'un programme (qu'on appelle aussi sa *sémantique*).

Attributs

- Des exemples typiques d'attributs sont:
 - Le *type* d'une expression
 - Le *symbole* produit par une déclaration
 - La *table des symboles* (ou *portée*) produit par un ensemble de déclarations.
- Les tables des symboles sont souvent représentées par des variables globales au lieu d'un ensemble d'attributs.
- Voilà une différence entre la théorie et la pratique !
- Il est toutefois important de s'assurer que la pratique ne détruit pas les concepts théoriques. Les tables des symboles peuvent être vues comme un attribut, mais on choisit une représentation centralisée plus performante.

Spécification de la grammaire contextuelle de misc

P = Program { D } E(te)

Crée une portée pour le programme entier, et l'utilise pour analyser les déclarations {D} et l'expression principale E.

D = FunDecl ident { F }({ta}) T(tt) E(te)

Analyse les arguments {F} dans une nouvelle portée, imbriquée dans la portée courante. Ajoute un nouveau symbole dans la portée courante, de nom ident et de type tf. Analyse ensuite le corps E dans la portée des arguments.

F(t) = Formal ident T(tt)

Ajoute un nouveau symbole dans la portée courante, de nom ident et de type tt.

T(t) = UnitType
 | IntType
 | ListType T(tt)
 | FunType { T }({ts}) T(tt)

Martin Odersky, LAMP/DI

7

Spécification de la grammaire contextuelle de misc (2)

S = VarDecl ident T(tt) E(te)

Analyse l'expression E dans la portée courante. Ajoute un nouveau symbole dans la portée courante de nom ident et de type tt.

| While E(tc) E(te)

| Exec(te)

E(t) = Block { S } E(t1)

Analyse {s} et E dans une nouvelle portée, imbriquée dans la portée courante.

| UnitLit

| IntLit int

| NilLit

| ...

Martin Odersky, LAMP/DI

8

Spécification de la grammaire contextuelle de misc (3)

```
E(t) = ...  
  | Ident ident  
  
  | Assign ident E(t1)  
  
  | FunCall E(tf) { E }({ta})  
  
  | If E(tc) E(t1) E(t2)  
  
  | ...
```

Martin Odersky, LAMP/DI

9

Spécification de la grammaire contextuelle de misc (4)

```
E(t) = ...  
  | Operation CmpOp E(t1) E(t2)  
  | Operation IntOp E(t1) E(t2)  
  | Operation Cons E(t1) E(t2)  
  
  | Operation Head E(t1)  
  | Operation Tail E(t1)  
  | Operation IsEmpty E(t1)
```

CmpOp= Eq | Ne | Lt | Le | Gt | Ge

IntOp= Add | Sub | Mul | Div | Mod

Martin Odersky, LAMP/DI

10

Le Sous-Typage

- Nouvelle Relation $s \prec: T$
 - s est un *sous-type* de T , ou s est *compatible* avec T .

- Règles:

$$T \prec: T$$

$$\text{bottom} \prec: T$$

$$\frac{S \prec: T \quad T \prec: U}{S \prec: U}$$

$$\frac{S \prec: T}{\text{List}[S] \prec: \text{List}[T]}$$

- Exemple: `List[bottom] <: List[Int]`

Martin Odersky, LAMP/DI

11

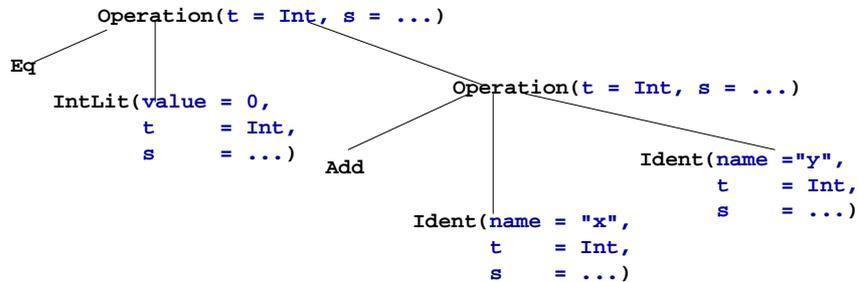
Comment obtenir un compilateur à partir d'une spécification

- Au lieu de *deviner* les attributs et de *vérifier* ensuite qu'ils satisfont les règles d'attribution, on doit les *calculer*.
- Les attributs sont en général calculés à partir de la valeur d'autres attributs.
- Important : les attributs doivent être affectés une et une seule fois !
- Les attributs peuvent être distingués en fonction de la manière dont ils « traversent » l'arbre syntaxique.
 - Certains le traversent de bas en haut - ils sont dits *synthétisés*.
 - Certains le traversent de haut en bas - ils sont dits *hérités*.
- Dans un compilateur, les attributs synthétisés sont représentés par les type de retour (ou paramètres de sortie en C/C++) des visiteurs.
- Les attributs hérités sont représentés par les paramètres (d'entrée) des visiteurs.

Martin Odersky, LAMP/DI

12

Exemple



- s est hérité
- t est synthétisé
- name, value sont intrinsèques

Martin Odersky, LAMP/DI

13

Représentation des attributs

- Certains attributs doivent être présents dans des phases ultérieures de la compilation - ils sont dits *persistants*.
- D'autres attributs ne sont utiles que lors de la vérification de type - ils sont dits *temporaires*.
- Les attributs persistants peuvent être stockés comme champs supplémentaires dans l'arbre syntaxique.
- Les attributs temporaires sont des paramètres et des résultats des méthodes des visiteurs :
 - Synthétisé, temporaire = résultat de visiteur.
 - Hérité, temporaire = paramètre de visiteur.
- Certains attributs peuvent aussi être représentés par des variables globales. Cela est plus simple si ces attributs changent peu souvent.

Martin Odersky, LAMP/DI

14

Exemple

```

E(t,s) = Operation IntOp E(t1, s1) E(t2, s2)
      | Operation CmpOp E(t1, s1) E(t2, s2)
      | Ident name (sym)
      | ...
    
```

- Les attributs suivants existent :
 - **t** : (type) synthétisé, persistant
 - **sym** : (symbole) intrinsèque, persistant
 - **s** : (portée) hérité, temporaire
- On a donc besoin pour l'analyse d'un visiteur structuré de la manière suivante.

Martin Odersky, LAMP/DI

15

Exemple (2)

```

public class Analyzer
  implements Tree.Visitor
{
  private Scope scope;

  public Analyzer(Scope scope) {
    this.scope = scope;
  }

  public Type analyze(Tree tree,
                     Scope scope)
  {
    {
      tree.apply(new Analyzer(scope));
      return tree.type;
    }
  }

  public void caseOperation(Operation tree){
    if (tree.operator == NEG) {
      Type t = analyze(tree.left, scope);
      if (t.tag != INT) {
        if (t.tag != BAD)
          error(tree.pos,
                "expected integer, " +
                "found " + Tree.type);
      }
      tree.type = Type.intType;
    }
    ...
  }

  public void caseIdent(Ident tree) {
    tree.sym = scope.lookup(tree.name);
    if (tree.sym == null) {
      error(tree.pos,
            tree.name + "undefined");
      tree.type = Type.badType;
    } else {
      tree.type = sym.type;
    }
  }
  ...
}
    
```

Martin Odersky, LAMP/DI

16

Optimisations :

1. La création d'un visiteur pour chaque nœud visité prend du temps.
On a meilleur temps de réutiliser le visiteur courant.

```
public Type analyze(Tree tree, Scope scope) {
    Scope scope1 = this.scope;
    this.scope = scope;
    tree.apply(this);
    this.scope = scope1;
    return tree.type;
}
```

2. L'attribut `scope` ne varie pas fréquemment.
⇒ on peut utiliser une méthode d'analyse plus efficace qui ne fait pas la sauvegarde/changement/restauration de la portée.

```
public Type analyze(Tree tree) {
    tree.apply(this);
}
```

Martin Odersky, LAMP/DI

17

Optimisations (2)

3. On doit souvent comparer deux types et signaler une erreur s'ils ne sont pas compatibles.

⇒ On peut créer une fonction `checkType` qui se charge de faire cela

```
public Type checkType(int pos, Type found, Type required) {
    if (found.isSubType(required)) return found;
    error(pos, "type error: " +
        "expected " + required + ", " +
        "found " + found);
    return Type.badType;
}
```

Dès lors `caseOperation` peut s'écrire de manière plus concise.

```
public void caseOperation(Operation tree) {
    if (tree.operator == NEG) {
        checkType(tree.pos, analyze(tree.left), Type.intType);
        tree.type = Type.intType;
    }
    ...
}
```

Martin Odersky, LAMP/DI

18

Spécification complète de la grammaire contextuelle de misc (2)

```

S = VarDecl ident T(tt) E(te)      te <: tt
  Analyse l'expression E dans la portée courante. Ajoute un
  nouveau symbole dans la portée courante de nom ident et de
  type tt.
  | While E(tc) E(te)              tc = Int
  | Exec(te)

E(t) = Block { S } E(t1)           t = t1
  Analyse {s} et E dans une nouvelle portée, imbriquée dans la
  portée courante.
  | UnitLit                        t = Unit
  | IntLit int                     t = Int
  | NilLit                         t = List[bottom]
  | ...
  
```

Martin Odersky, LAMP/DI

21

Spécification complète de la grammaire contextuelle de misc (3)

```

E(t) = ...
  | Ident ident                    sym = lookup(ident)
                                   t = sym.type
  | Assign ident E(t1)            sym = lookup(ident)
                                   sym.isVariable()
                                   t1 <: sym.type
                                   t = Unit
  | FunCall E(tf) { E }({ta})     tf = ({ts}) tt
                                   {ts}.length={ta}.length
                                   for i in 1..{ts}.length
                                   {ta}[i] <: {ts}[i]
                                   t = tt
  | If E(tc) E(t1) E(t2)         tc = Int
                                   t1 <: t
                                   t2 <: t
  | ...
  
```

Martin Odersky, LAMP/DI

22

Spécification complète de la grammaire contextuelle de misc (4)

`E(t) = ...`

```
| Operation CmpOp E(t1) E(t2)  t = t1 = t2 = Int
| Operation IntOp E(t1) E(t2)  t = t1 = t2 = Int
| Operation Cons E(t1) E(t2)   List[t1] <: t
                                t2 <: t
| Operation Head E(t1)         List[t] = t1
| Operation Tail E(t1)         t = t1 = List[t2]
| Operation IsEmpty E(t1)      t1 = List[t2]
                                t = Int
```

`CmpOp= Eq | Ne | Lt | Le | Gt | Ge`

`IntOp= Add | Sub | Mul | Div | Mod`