

Transformation de programmes et C++

Robert Anisko <robert@lrde.epita.fr>

Séminaire du LRDE, 30 Octobre 2002



Table des matières

Introduction	4
Outils	8
Projets.....	9
Analyse syntaxique : Sglr	10
Réécriture : Stratego	11
Architecture	12
Grammaire	13
Problèmes.....	14
Traitement de la grammaire.....	15

Table des matières

Chaîne de traitement	17
Parsing non-déterministe et filtrage	19
Exemple 1/2	20
Exemple 2/2	21
Chaîne de traitement	22
Filtres locaux ou spécifiques	23
Exemple complet : résolution générale des ambiguïtés	24
Filtre : espaces de noms (problème)	25
Filtre : déclarations sans déclarateurs (problème)	27
Autres filtres locaux	29

Table des matières

Analyse globale du programme	30
Conclusion	33

Introduction

Transformation de programmes

- Réécriture sur arbres.
- Contraintes sur les outils :
 - ▷ Analyse syntaxique.
 - ▷ Transformations.
 - ▷ Pretty-printing.

Applications

- Rénovation de programmes.
- Documentation, instrumentation de code.
- Optimisation (évaluation partielle, etc.).
- Traduction et compilation.
- Traitement automatique.

C++

- Bibliothèques génériques (Olena, Vaucanson) :
 - ▷ Généricité + performances \Rightarrow méta-programmation.
 - ▷ Code complexe, difficile à écrire.
 - ▷ Difficile à lire et à maintenir.
- Application : transformation de C++ classique ou étendu \rightarrow C++ avec méta-programmation.

Outils

Projets

ASF+SDF Meta-Environment

Centrum Voor Wiskunde en Informatica, Amsterdam (Generic Language Technology Project). Système de réécriture.

Stratego

Pacific Software Research Center, Portland, puis université d'Utrecht. Système de réécriture.

GPP

Pretty-printer générique.

XT

Collection d'outils de réécriture (inclus *Sglr*, *Stratego*, *GPP*, etc.).

Analyse syntaxique : Sglr

- Scanner-less generalized LR.
- Grammaires hors-contexte, pas de restrictions \Rightarrow pas de massage.
- Non-déterministe \Rightarrow grammaires ambiguës.
- Grammaires hors-contexte stables pour l'union \Rightarrow grammaires modulaires (SDF).
- Grammaires modulaires \Rightarrow extension facile des spécifications.

Réécriture : Stratego

- Transformations sur des arbres de syntaxe abstraite.
- Autorise également les transformations sur des arbres de parse.
- Stratégies de réécriture ; spécifications modulaires et réutilisables.
- Bibliothèque de règles et de stratégies très complète : parcours d'arbres génériques, types de données, interaction avec le système...

Architecture

- Rôle de la grammaire.
- Chaîne de traitement des programmes.

Grammaire

- Traduction vers SDF de la grammaire BNF du standard C++ ISO/IEC 98.
- Pas de massage.
- Syntaxe lexicale et hors-contexte, approximativement 520 productions au total.

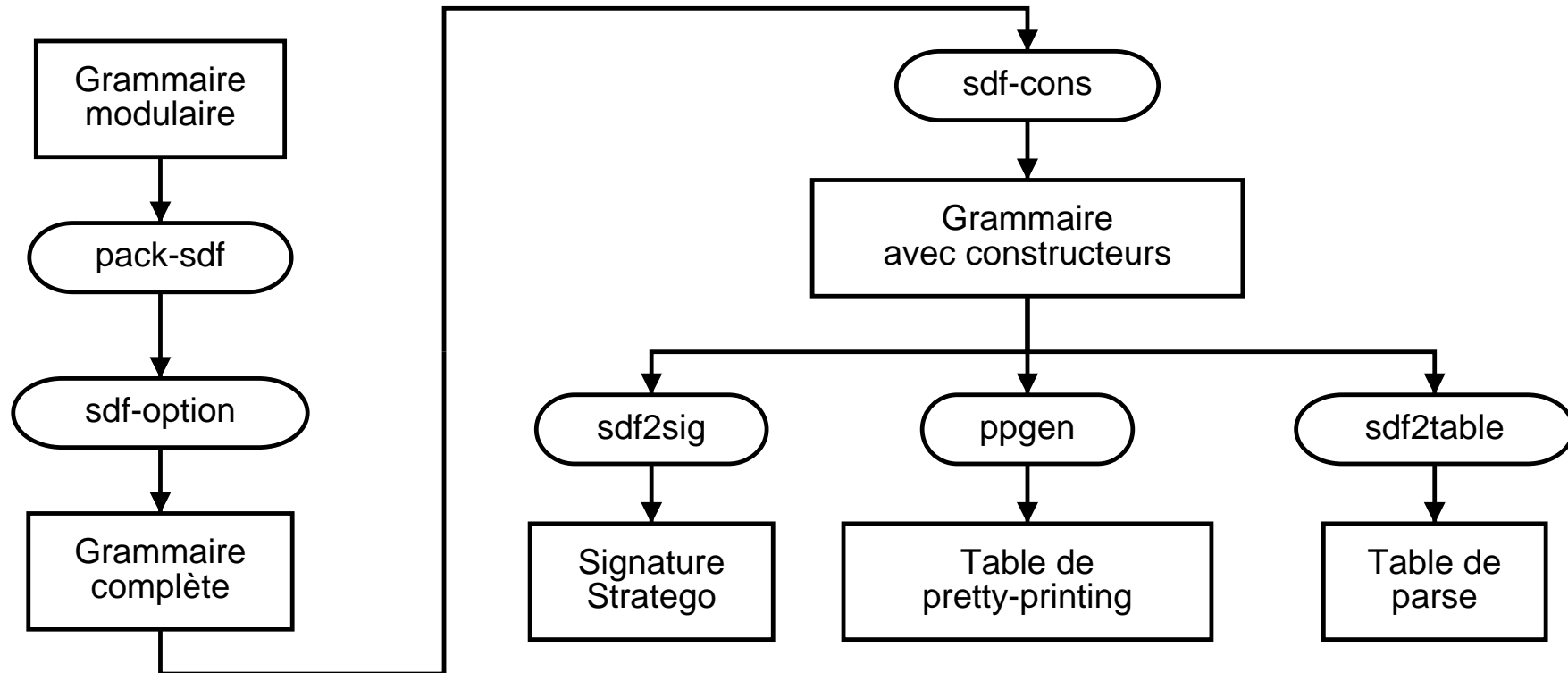
Problèmes

- Langage C++ pas hors-contexte.
- Vraie grammaire du langage → grammaire hors-contexte ambiguë, sur-ensemble du langage.
- Grammaire standard pas assez contraignante ⇒ ne rejette pas certains programmes incorrects.
- Certaines règles trop simples, permettent de réduire le nombre de productions.

Traitement de la grammaire

- Correction de certains défauts (`sdf-option`).
- Annotation de la grammaire avec des constructeurs (`sdf-cons`).
- Production d'une table de parse (`sdf2table`).
- Production de signatures Stratego (`sdf2sig`).
- Production d'une table de pretty-printing (`ppgen`).

Traitement de la grammaire



Chaîne de traitement

- Chaîne de traitement classique : analyse syntaxique puis transformation.
- Ici : parsing non-déterministe, retourne une forêt de parse \Rightarrow étape de filtrage pour réduire la forêt à un arbre (désambiguisation).

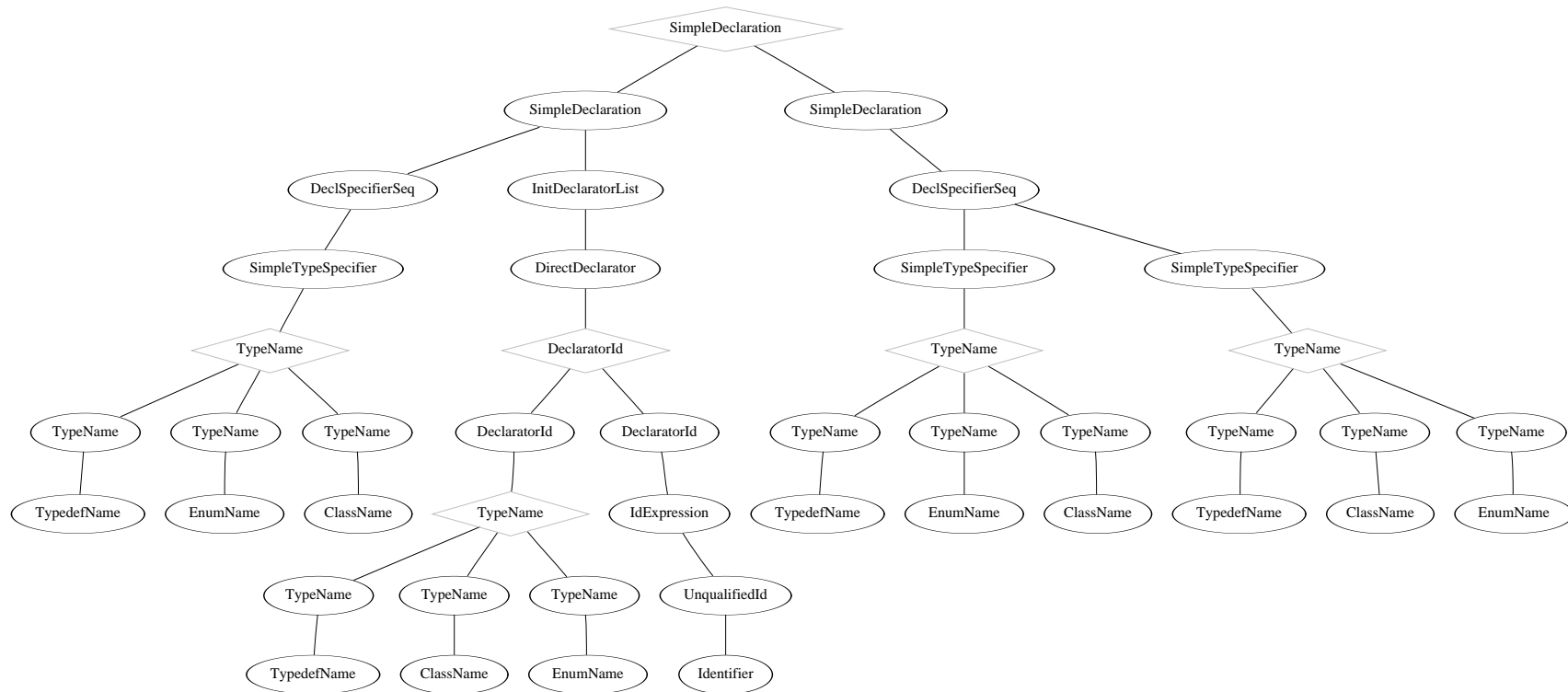
Chaîne de traitement

1. Parsing avec Sglr.
2. Filtrage de la forêt de parse.
3. Implosion de l'arbre de parse vers un arbre de syntaxe abstraite.
4. Transformation.
5. Pretty-printing avec Gpp.

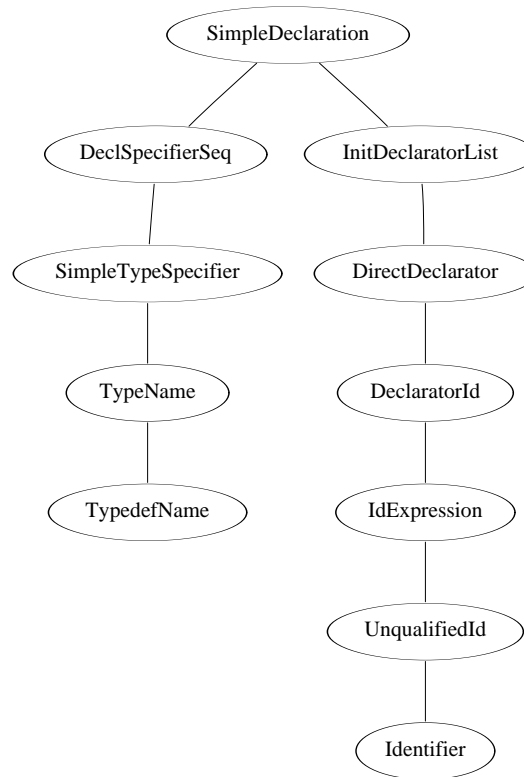
Parsing non-déterministe et filtrage

- Architecture : grammaire simple, parsing ambigu \Rightarrow post-traitement des forêts de parse.
- Traitement : filtres en Stratego, transformations sur des arbres de parse.

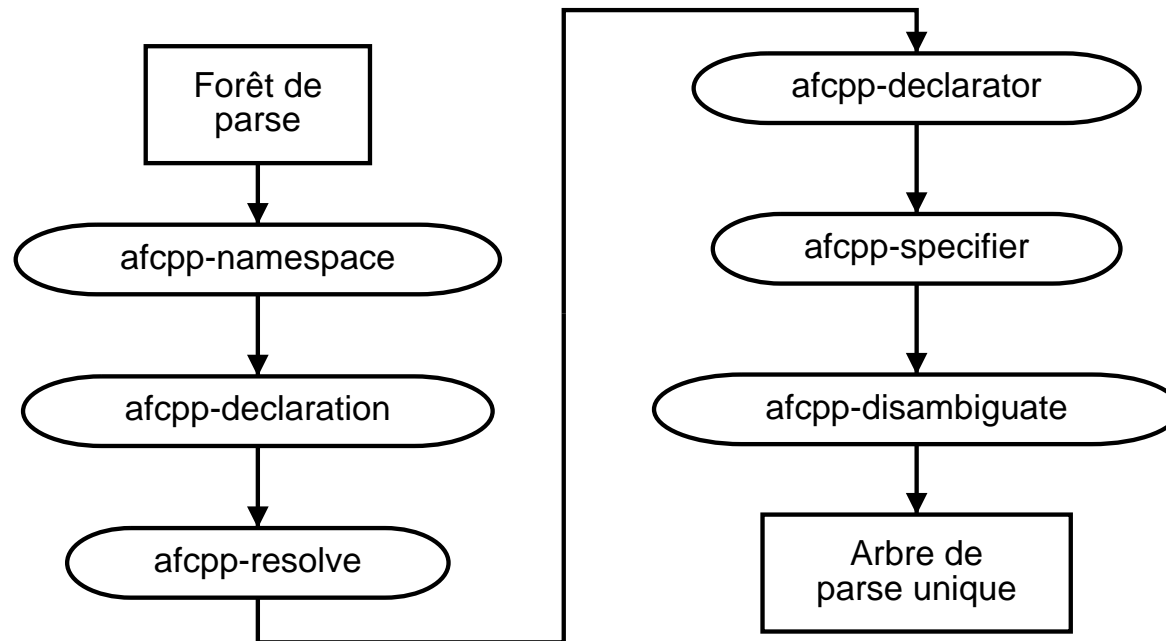
Exemple 1/2



Exemple 2/2



Chaîne de traitement



Filtres locaux ou spécifiques

- Traitement :
 - ▷ Ambiguïtés locales.
 - ▷ Éléments spécifiques du langage.
- But : réduire la forêt suffisamment pour permettre une analyse globale du programme.

Exemple complet : résolution générale des ambiguïtés

- Problème : certains filtres ne retirent pas directement les ambiguïtés.
- Noeud ambigu suffisamment réduit → retrait.

```
module Main
imports
  lib
  AsFix–Syntax
rules
  Resolve : amb ([a]) → a
           where < debug > “Resolved an ambiguity”
strategies
  main = iowrap (topdown (try (Resolve)))
```

Filtre : espaces de noms (problème)

- Problème : distinction entre la définition d'un namespace et son extension.

```
namespace foo { int a; } // Première définition du namespace foo.  
namespace bar { int a; } // Première définition du namespace bar.  
namespace foo { int b; } // Extension du namespace foo.
```

- Contraintes insuffisantes dans la grammaire hors-contexte.

<i>Identif</i>	→	<i>OriginalNamespaceName</i>
namespace <i>Identif</i> { <i>NmBody</i> }	→	<i>OriginalNamespaceDefinition</i>
namespace <i>OriginalNamespaceName</i> { <i>NmBody</i> }	→	<i>ExtensionNamespaceDefinition</i>

Filtre : espaces de noms (algorithme)

```
 $E \leftarrow \{\}$   
top-down traversal  
  for each ambiguous definition  $d$  of namespace  $n$  do  
    if  $n \in E$  then  
       $d$  is an extension of  $n$   
    else  
       $d$  is the original definition of  $n$   
       $E \leftarrow E \cup \{n\}$   
    end if  
  end for  
end top-down traversal
```

- Application très tôt dans la chaîne :
 - ▷ Permet de réduire considérablement la taille des forêts de parse.
 - ▷ Élimine les doublons.

Filtre : déclarations sans déclarateurs (problème)

- Déclaration : liste de qualificatifs + liste de déclarateurs.
- Problème : classification de certaines constructions impossible pendant l'analyse syntaxique \Rightarrow groupage ambigu dans certaines déclarations.

```
int foo = 0;      // Non ambigu.
int foo, bar;    // Non ambigu.
foo bar;         // Ambigu : ([foo bar], []) ou ([foo], [bar]).
typedef foo bar; // Ambigu : ([typedef foo bar], []) ou ([typedef foo], [bar]).
foo bar = 0;     // Non ambigu.
foo bar, baz;   // Non ambigu.
class A { };    // Non ambigu.
class A { } a;  // Ambigu : ([class A { } a], []) ou ([class A { }], [a]).
```

Filtre : déclarations sans déclarateurs (algorithme)

- Critère d'élagage : pas de déclarations avec une liste de déclarateurs vide.
- Faux dans le cas général.
- Les déclarations pour lesquelles ce critère ne se vérifie pas ne sont pas ambiguës.

Autres filtres locaux

- Déclarations :
 - ▷ Déclarateurs ambigus.
 - ▷ Groupage des qualificateurs.
- Succès des filtres locaux \Rightarrow informations suffisantes pour faire l'analyse sémantique du programme.

Analyse globale du programme

- Forêt de parse suffisamment réduite : déclarations lisibles.
- Traitement en deux passes :
 1. Lecture des déclarations, construction d'un environnement pour chaque espace de nom.
 2. Utilisation des informations collectées, processus de vérification récursif : pour chaque noeud ambigu, conserver la branche correcte.

Analyse globale du programme (algorithme)

{build environments}

top-down traversal

for each definition of namespace or class n **do**

$E_n \leftarrow \{\}$

end for

for each declaration d of symbol s **do**

s is declared in namespace n

s is of kind k

$E_n \leftarrow E_n \cup \{s : k\}$

end for

end top-down traversal

{*disambiguate*}

top-down traversal

for each ambiguous node $a(a_1, a_2, \dots, a_n)$ **do**

keep a_i such as *disambiguate*(a_i) is successful

end for

for each symbol s seen with kind k **do**

find namespace n in which s is declared

if $E_n \vdash s : k' \neq k$ **then**

fail

end if

end for

for each local declaration d of symbol s **do**

s is declared in namespace n

s is of kind k

$E_n \leftarrow E_n \cup \{s : k\}$

end for

for each scope in namespace n **do**

save and restore E_n properly

end for

end top-down traversal

Analyse globale du programme

- Succès de l'analyse globale :
 - ▷ Fin de la désambiguïsation.
 - ▷ Résultat : arbre de parse non ambigu, unique interprétation correcte du texte entré.
- Echec du filtrage : programme incorrect.
- Succès du filtrage : aucune garantie que le programme soit correct.

Conclusion

- Résultat :
 - ▷ Outils disponibles pour le C++.
 - ▷ Transformations sur un langage complet et non hors-contexte.
- Perspectives :
 - ▷ Filtrage complet (templates, etc.).
 - ▷ Autres problèmes : pré-processeur, commentaires.
 - ▷ Vérification des programmes.
 - ▷ Application des outils aux bibliothèques génériques.