AnalysedeDonnéesStructurées-Cours2

RalfTreinen

UniversitéParisDiderot
UFRInformatique
LaboratoirePreuves,ProgrammesetSystèmes

treinen@pps.univ-paris-diderot.fr

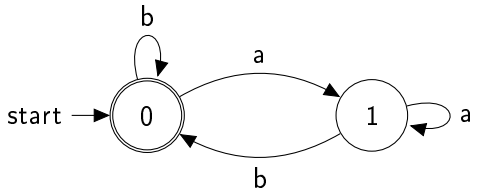
28janvier2015

AutomatesDéterministes

- Étantdonnéunalphabetfini Σ.
- Unautomateestuntuple (Q, q_o, F, Δ) où
  - Q estunensemblefinid'états
  - $q_0 \in Q$ estl'étatinitial
  - $F \subseteq Q$ estl'ensembled'étatsacceptants
  - $\Delta\colon Q \times \Sigma \to Q$ lafonctiondetransition.

Exemple($\Sigma = \{a, b\}$)

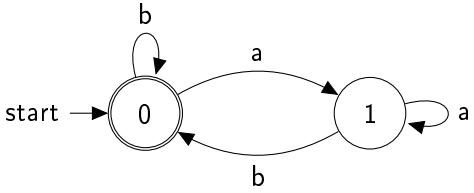
- (Q, q_0, F, Δ) où
  - $Q = \{0, 1\}$
  - $q_0 = 0$
  - $F = \{0\}$
  - $\Delta(0, a) = 1 \quad \Delta(0, b) = 0$
    $\Delta(1, a) = 1 \quad \Delta(1, b) = 0$
- Représentationgraphique:



Exécutionsurunmot

- Définitiond'unefonctiond'exécution $\Delta^*\colon Q \times \Sigma^* \to Q$ par récurrence:
  - $\Delta^*(q, \epsilon) = q$
  - $\Delta^*(q, xw) = \Delta^*(\Delta(q, x), w)$ où $x \in \Sigma$
- Anautomate $A = (Q, q_o, F, \Delta)$ *accepte* lemot $w \in \Sigma^*$ ssi $\Delta^*(q_o, w) \in F$.
- $\mathcal{L}(A) = \{w \in \Sigma^* \mid \Delta^*(q_0, w) \in F\}$
- Unlangage L est *régulier* quandilexisteunautomate A tel que $L = \mathcal{L}(A)$.

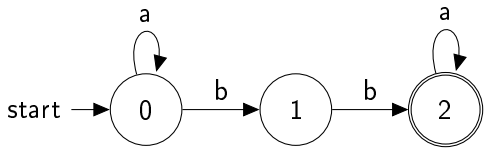
Sur l'exemple



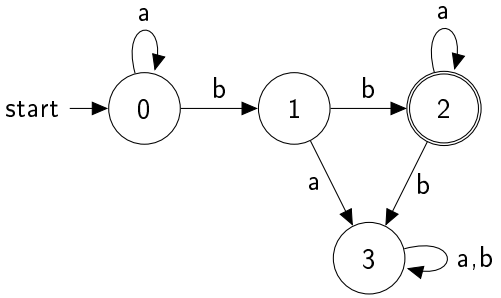
- $\Delta^*(q_0, bbab) = 0 \in F$: mot accepté
- $\Delta^*(q_0, bababa) = 1 \notin F$: mot pas accepté
- $\mathcal{L}(A) = \{\epsilon\} \cup \{wb \mid w \in \Sigma^*\}$

Raccourci : Automate avec une fonction Δ partielle

-

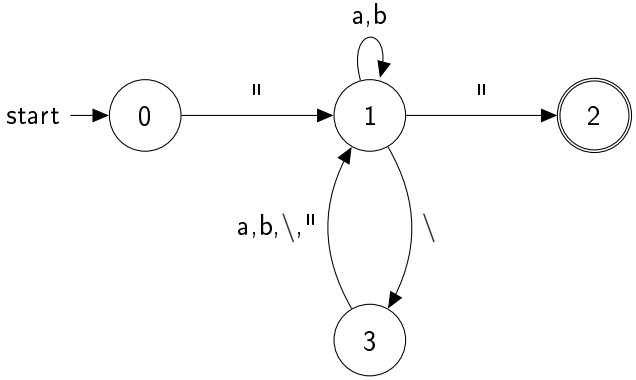


- Raccourci pour :



Exemple : un automate pour les strings de Java

- Simplification : $\Sigma = \{a, b, \backslash, "\}$
- Expression rationnelle vue au cours 1 : $"(a \mid b \mid \backslash.)*"$
- Automate :



Vu dans le cours *Langages et Automates* du S3

- Automates *non*-déterministes
- Automates avec ϵ-transitions
- Algorithme de *déterminisation*
- Algorithme de *minimisation*
- Traduction d'un automate en expression régulière
- Traduction d'une expression régulière en automate
- Un langage est rationnel si et seulement s'il est régulier.

Une classe Java très simple pour un automate donné

- Représenter les états par des entiers $0, 1, \ldots, nbe - 1$
- Représenter l'ensemble des états acceptants par un vecteur de booléens
- Représenter les symboles de l'alphabet par des entiers $0, \ldots, nbs - 1$
- Représenter la fonction de transition par un tableau à deux dimensions.
- On utilise la valeur -1 pour l'état poubelle implicite.

Implémentation naïve I

```
public class Abba {

    static int nbe=3;    /* nombre d'etats */
    static int nbs=2;    /* nombre de symboles */
    static int q0=0;     /* etat initial */
    static boolean[] accepting = {false, false, true};
    static int[][] delta = {{0,1}, {-1,2}, {2,-1}};

    public static int int_of_char(char c) {
        if (c=='a') {
            return 0;
        } else {
            return 1;
```


Implémentation naïve II

```
        }
    }

    public static void main(String[] args) {
        int q=q0;
        for(int i=0; i args[0].length(); i++) {
            q=delta[q][int_of_char(args[0].charAt(i))];
            if (q==-1) {break;}
        }
        if ( q > -1 && accepting[q] ) {
            System.out.println("accepted");
        } else {
            System.out.println("not accepted");
```


Implémentation naïve III

```
        }
    }
}
```

Un problème en pratique

- La taille du tableau des transitions est le nombre d'états $\times$ le nombre de symboles.
- Gourmand en mémoire quand l'alphabet est très grand (Unicode : $>$ 1.000.000 caractères).
- En pratique il y a très peu de flèches qui partent d'un état, toutes étiquetées par une classe de symboles.
- Il convient de trouver une implémentation plus efficace en mémoire, à l'aide des classes de caractères.

Parenthèse : recherche d'un mot dans un texte

- On veut savoir si un mot de longueur r (le motif) apparaît dans un texte de longueur n.
- Cas d'application typique : $n >> r$.
- L'algorithme naïf qu'on a vu en IF1/IP1 a une complexité $n * r$.

L'algorithme naïf I

```
public class GrepNaif {
    public static void main(String[] args) {
        String motif = args[0];
        String texte = args[1];
        int r = motif.length();
        int n = texte.length();
        boolean found = false;

        for(int i=0; i n-r+1; i++) {
            for(int j=0; j r; j++) {
                if (texte.charAt(i+j) != motif.charAt(j)) {
                    break;
                }
                if (j==r-1) {
                    found=true;
```


L'algorithme naïf II

```
                }
            }
            if (found) {
                break;
            }
        }

        if (found) {
            System.out.println("accepted");
        } else {
            System.out.println("not accepted");
        }
    }
}
```

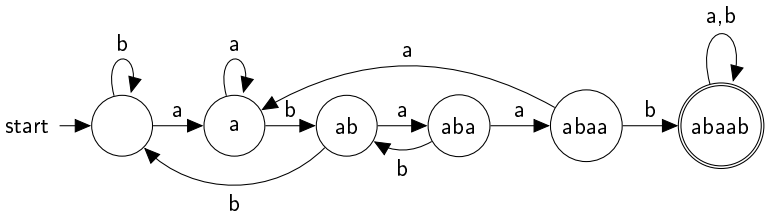
Recherche d'un mot dans un texte : l'idée

- Les automates permettent un algorithme plus efficace !
- Idée : on construit l'automate qui correspond à l'expression régulière $\Sigma * r \Sigma *$.
- Cet automate s'exécute sur le texte avec une complexité de n : Chaque caractère du texte est traité une seule fois.
- Construction de l'automate : on peut utiliser la traduction générique des expressions régulières vers des automates ...
- ... ou faire une construction directe.

La construction directe de l'automate

- Les états sont les *préfixes* du motif r.
- Exemple : $r = abaab$.
  Préfixes : ϵ, a, ab, aba, $abaa$, $abaab$.
- Signification : l'état signifie le *préfixe le plus long* de r qui est un *suffixe* de la partie du texte vue.
- État initial : ϵ.
  États acceptants : $\{r\}$
- Construction de Δ ? Voir l'exemple.

L'automate pour $\Sigma * abaab \Sigma *$



Construction de l'automate

- Un algorithme efficace pour la construction de l'automate, pour un motif r quelconque ?
- C'est le célèbre algorithme de *Knuth, Morris, et Pratt*.
- Complexité linéaire dans la taille du motif.
- Conséquence : Complexité $n + r$ pour la recherche d'un motif dans un texte (au lieu de $n * r$ avec l'algorithme naïf).
- Voir un cours d'Algorithmique.

L'objectif de l'analyse lexicale

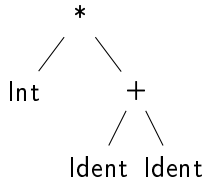
- Découper un texte d'entrée en une séquence de *lexèmes*, et les représenter par des *jetons* (*tokens* en anglais)
- À la base : Classification des lexèmes qui peuvent paraître dans un texte d'entrée, à l'aide des expressions régulières.
- La phase suivante de l'analyse (l'analyse syntaxique, voir plus tard) va travailler sur le résultat de ce découpage : il s'agit d'une *abstraction* du texte d'entrée.

Exemple

- Texte d'entrée :

(	7	5	6	e	2		*		(	e	5	e	7		+		v	a	l	e	u	r	2	)	)

- Jetons :
  PARG INT MULT PARG IDENT PLUS IDENT PARD PARD
- Résultat de l'analyse syntaxique (voir plus tard) :



Définition des catégories lexicales

Sur l'exemple des expressions arithmétiques (on utilise \ comme symbole d'échappement) :

- INT : [0..9]+(e[0..9]+) ?
- IDENT : [a..zA..Z][a..zA..Z0..9]*
- PARG : \(
- PARD : \)
- MULT : *
- PLUS : \+

Jetons avec arguments

- En réalité, on veut aussi garder certaines informations avec les jetons, comme la valeur d'une constante entière, ou le nom d'un identificateur.
- Certains jetons doivent donc avoir un argument :
  - IDENT(string)
  - INT(int) (c'est bien `int` et pas `string` !)
- Séquence des jetons obtenue sur l'exemple :
  PARG INT(75600) MULT PARG IDENT("e5e7") PLUS IDENT("valeur2") PARD PARD

Ignorer des informations pas pertinentes

L'analyse lexicale sert aussi à faire abstraction de certaines informations dans le texte d'entrée qui ne sont pas pertinentes pour l'analyse du texte. Souvent il s'agit de :

- Les espaces : sont utiles pour indiquer la fin d'un mot. Les espaces sont utiles *pour* l'analyse lexicale, mais une fois le découpage fait on peut les oublier.
- Les commentaires : souvent l'analyse lexicale vérifie l'écriture correcte des commentaires, mais ne les représente pas dans sa sortie.

Exemple

Différents textes d'entrée qui peuvent donner la même séquence de jetons :

- `34 * (x + y)`
- `34*(x+y)`
- `34     *(     x+   y)`
- `34 * (x+y) (* Ceci est un comment ire *)`

Quelle information retenir dans les jetons

- On retient dans les jetons seulement l'information qui est utile pour la suite.
- La distinction entre information utile/inutile dépend de l'application.
- Par exemple : Les commentaires peuvent être utiles à retenir pour certaines applications.
- Il peut être utile de conserver avec les jetons aussi des informations de *localisation* : nom du fichier source, numéro de ligne, numéro de colonne.

Résoudre les ambiguïtés

- L'analyse lexicale va, pour produire le jetons suivant, chercher un *préfixe* du reste du texte qui correspond à une des catégories lexicales, et construire le jeton correspondant.
- Il y a deux sources d'ambiguïtés :
  - Des préfixes de longueurs différents peuvent être reconnues
  - Les expressions régulières peuvent avoir une intersection non vide

Préfixes de longueur différentes reconnues

Exemple :

- Catégorie lexicale :
  - IDENT : `[a..z]+`
- Début du texte d'entrée :
  `xyz`
- Plusieurs possibilités de découpage :
  1. `IDENT("x") IDENT("y") IDENT("z")`
  2. `IDENT("xy") IDENT("z")`
  3. `IDENT("x") IDENT("yz")`
  4. `IDENT("xyz")`
- La règle normale est : on cherche le préfixe *maximal*.

Plusieurs expressions régulières s'appliquent

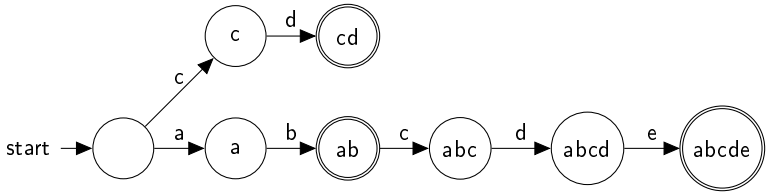
Exemple :

- Catégories lexicales :
  - PUBLIC : `public`
  - IDENT : `[a..z]+`
- Début du texte d'entrée :
  `public public tion`
- Plusieurs possibilités de découpage :
  1. `IDENT("public") IDENT("publication")`
  2. `PUBLIC IDENT("publication")`
- La règle normale est : à longueur égale du mot reconnu, c'est la première expression régulière qui gagne.

Exécution de l'automate pour le recherche d'un préfixe maximal

Exemple (artificiel) :

- Catégorie lexicales :
  - AB : `ab`
  - CD : `cd`
  - ABCDE : `abcde`
- Automate :



Exécution de l'automate pour le recherche d'un préfixe maximal

- Si on cherche le préfixe le plus *court* reconnu on doit s'arrêter dès qu'on arrive dans un état acceptant.
- Si on cherche le préfixe le plus *long* reconnu on doit continuer à lire tant que possible, et quand on passe par un état acceptant :
  - mémoriser l'état acceptant ;
  - mémoriser la position dans le mot d'entrée

  Si l'automate ne peut plus continuer : remettre le pointeur de lecture dans l'entrée à la position où on a vu le dernier état acceptant.

AnalysedeDonnéesStructurées-Cours3

RalfTreinen

UniversitéParisDiderot
UFRInformatique
LaboratoirePreuves,ProgrammesetSystèmes

treinen@pps.univ-paris-diderot.fr

4février2015

Objectifdel'analyselexicale

- Lireletexted'entrée,etfaireunpremiertraitementenvue d'unesimplificationpourlesétapessuivantes:
- *Découpage* del'entréeen *lexèmes* (desmotsélémentaires)
- *Classer* leslexèmesidentifiés,créationdejetons
- *Interpréter* leslexèmesquandpertinent,parexemple transformerunesuitedechiffresenunentier
- *Abstraire* l'entrée:ignorerdesdétailsnonpertinentspourla suite(espaces,commentaires. *:::*)

Spécificationd'uneanalyselexicale

- Définitiondescatégorieslexicalesdifférentes:expressions régulières
- Définitiondelastratégie:chercherlemotlepluslongoule pluscourt,commentrésoudredesambiguïtés
- Définirletypedesjetonsproduitsavecleursarguments éventuels.

Implémenteruneanalyselexicale

- Soit *écrireunprogramme* (Javaouautre)àlamain,basésur unautomatefini:discutéauderniercours. Lespremierscompilateursétaienteffectivementécritsdecette façon(compilateurdulangageFORTRAN,Backusetal. 1957:18personnes-années).
- Soitfaire *engendrer* unanalyseurlexicaleàpartird'une spécification(cecours). C'estlatechniqueutiliséepourl'écrituredescompilateurs modernes.

L'interfacedel'analyseurlexicale

- Onpourraitimaginerquel'analyselexicalevacréeruneliste Javaavectouslesjetonscrééslorsdel'analyse.
- Problème:cettelisterisqued'êtretrèslongue.
- Normalement,laphasesuivantedel'analyseaseulement besoindelirelesjetonsunefoisdansl'ordre.
- Pourcesraisons,l'analyselexicalecréelesjetonsunaprès l'autre *àlademande* :fonctionquirenvoielejetonsuivant.

Différentsgénérateurs

- Existentpourpresquetousleslangagesdeprogrammation.
- Lepremiergénérateurétait *lex*,publiéen1975parMikeLesk etEricSchmidt.Engendreducodeen C.
- Successeur: *flex*,1987.
- Lesgénérateursmodernessontsouventissusdeflex.Nous utilisonsiciungénérateurpourJava: *jflex*.
- Lesgénérateurspourdesautreslangagesdeprogrammation sonttrèssimilaires.

Lecodeengendrédanslecasdejflex

- Uneclassepourl'analyseurlexicale,lenomdelaclassepeut êtredéfinidanslaspécification(dansnosexemples: `Lexer`).
- Lacréationd'unobjetdecetteclasse(unanalyseur)prenden argumentunobjetquireprésentele *flotd'entrée* ,parexemple unfichier,oul'entréestandard.
- Ilyuneméthodepourdemanderlejetonsuivant.Lenomde cetteméthode,etletypedesjetons,peuventêtredéfinisdans laspécification.

Laspécification

- Troisparties,séparéespardeslignes `%%` :
  - codeutilisateur
  - optionsetdéclarations
  - règleslexicales
- Codeutilisateur:
  - copiésimplementaudébutdufichierengendré(avantla définitiondelaclasse)
  - partiesouventvide(saufcommentaires,et `import...`)

La partie *Options et Déclarations*

- Options : commencent avec le symbole `%`. Parmi les options les plus importantes :
  - `%class` *nom* : donne le nom de la classe engendrée.
  - `%public` : la classe engendrée est publique
  - `%type` *t* : le type de résultat de la fonction `yylex`.
  - `%unicode` : accepte des caractères Unicode en entrée de l'analyse lexicale (recommandé)
  - `%line` : compte les lignes pendant l'analyse lexicale (disponible en `yyline`).
  - `%column` : compte colonnes pendant l'analyse lexicale (disponible en `yycolumn`).
  - `%state` *s* : Déclaration de l'état *s* (voir plus tard)

La partie *Options et Déclarations*

- Code entre `%{` et `%}` (peut être sur plusieurs lignes) :
  - copié au début de la classe engendré
  - ce code a donc accès aux champs de la classe (par exemple, `yyline`, `yycolumn`)
- Code entre `%eofv 1{` et `%eofv 1}` : code exécuté quand l'analyse lexicale arrive à la fin de l'entrée (défaut : `null`).

La partie *Options et Déclarations*

- Macros : système de définitions d'expressions régulières
  - mettre les mots entre apostrophes `"` et `"`
  - pour utiliser une expression régulière préalablement définie, par exemple du nom *r* : `{r}`.
  - classes de caractères, par exemple `[a-z]`
  - quelques classes de caractère prédéfinies, par exemple `[:letter:]`, `[:digit:]`, `[:uppercase:]`, `[:lowercase:]`.

La partie *Règles Lexicales*

- Séquence de
  expression-régulière { *code-java* }
- dans le cas le plus simple, le code java a est un `return...`
- Règles d'exécution : on cherche le lexeme le plus long possible, et on applique l'action de la première expression régulière qui s'applique.

Lepremierexemple

- Analyselexicalepourdesexpressionsarithmétiquescommevu ladernièrefois.
- Petitedifférenceaupremierexemple:lesentiersne contiennentpasd'exposant.
- Définitiondesclassespourles Symboles(typedejetons),puis pourlesjetonséventuellementavecdesarguments.
- Lefichierdespécificationpour `jflex`.
- Unpetitprogrammeprincipalpourtester.

Fichier Sym.java

```
public enumSym{
    INT,IDENT,PARG,PARD,MULT,PLUS;

    public Stringstr(){
        switch (this){
        case INT: return "INT";
        case IDENT: return "IDENT";
        case PARG: return "PARG";
        case PARD: return "PARD";
        case MULT: return "MULT";
        case PLUS: return "PLUS";
        default : return "impossible";
        }
    }
}
```


Fichier Token.java I

```
class Token{
    protected Symsymbol;
    public Token(Syms){
        symbol=s;
    }
    public Symsymbol(){
        return symbol;
    }
    public Stringstr(){
        return (symbol.str());
    }
}

class StringToken extends Token{
    private Stringvalue;
    public StringToken(Symc,Strings){
        super (c);
        value=s;
```


Fichier Token.java II

```
    }
    public Stringstr(){
        return (symbol.str()+'('+value+')');
    }
}

class IntToken extends Token{
    privateint value;
    public IntToken(Symc, int i){
        super (c);
        value=i;
    }
    public Stringstr(){
        return (symbol.str()+'('+value+')');
    }
}
```

Fichier arith.flex I

```
%%
%public
%class Lexer
%unicode
%typeToken

%{
    private Tokentoken(Symtype){
        returnnew Token(type);
    }
    private StringTokentoken(Symtype,Stringvalue){
        returnnew StringToken(type,value);
    }
    private IntTokentoken(Symtype, int value){
        returnnew IntToken(type,value);
    }
%}
```


Fichier arith.flex II

```
EspaceChar=[\n\r\f\t]
Ch =[0 −9]
Le =[a −zA−Z]

%%
{Ch}+ { return token(Sym.INT,
                Integer.parseInt(yytext()));}
{Le}({Le}|{Ch}) * { return token(Sym.IDENT,yytext());}
"(" { return token(Sym.PARG);}
")" { return token(Sym.PARD);}
"*" { return token(Sym.MULT);}
"+" { return token(Sym.PLUS);}
{EspaceChar}+ {}
```


FichierTest.java

```
import java.io. *;

class Test{

    public static void main(String[]args) throws Exception{
        Fileinput= new File(args[0]);
        Readerreader= new FileReader(input);
        Lexerlexer= new Lexer(reader);
        Token t;
        do {
            t=lexer.yylex();
            if (t!= null ){System.out.println(t.str());}
        } while (t!= null );
    }
}
```


L'automatecréé

- Créationdesclassesdecaractères:touslescaractèresquine sontjamaisdistinguésparlesexpressionsrégulièressont groupésdanslamêmeclasse.
- Lesclassescréésdoiventêtredisjointes.
- Exemple:expressionsrégulières:
  `"end"`
  `[ -z]*`
- Quatreclassesdecaractèresdisjointes: `[e]`, `[n]`, `[d]`, `[ -cf-mo-z]`

L'automatecréé

- Créationd'unautomatenon-déterministepourl'uniondetouteslesexpressionsrégulières.
- Déterminiserl'automate(etéliminerles -transitions).
- Minimiserl'automate.
- Onpeutdemanderà `jflex` demontrercestroisautomates(option `-dot`,visualiserlesautomatesavec `xdot` parexemple)

Lesétatsdel'analyseurlexical

- Pardéfaut(commesurlepremierexemple),votreanalyseurlexicalaunseulétat.
- Pourenavoirplusieurs:
  - lesdéclareràl'aidede `%state` (saufYYINITIAL)
  - mettretouteslesrèglesdanslecontexted'unétat
  - danslesactions:changerd'étatàl'aidede `yybegin`.

Pourquoiutiliserplusieursétats?

- Unpremierexemplesontlescommentaires:avecuneexpressionrégulièrescomme `"/*".*"*/"` onaunproblèmequandilyaplusieurscommentairesdansletexte(pourquoi?)
- Danscecasonveutonfaittrouverlemot *lepluscourt* décritparl'expressionrégulière.Celapeutêtresimuléenutilisantdeuxétats.

Reconnaîtrelescommentaires(simplifié)

```
%%
%typeToken
 EspaceChar=[\n\r\f\t]
 Letter=[a-zA-Z]
%stateINCOMMENT
%%
<YYINITIAL>{
   {Letter}+ {  return token(Sym.IDENT,yytext());}
   {EspaceChar}{}
   "/*" {yybegin(INCOMMENT);}
}
<INCOMMENT>{
   "*/"{yybegin(YYINITIAL);}
   .{}
}
```

Reconnaîtrelescommentaires

- `YYINITIAL` estl'étatpardéfaut
- Ilestcrucialqueladernièrerègles'appliqueàunmotde longueur1seulement.

Exemple:découperunmotenplusieursparties

- Retourànotrepremierexemple:onsouhaitemaintenantaussi reconnaîtredesentiersavecexposant(756e2,parexemple).
- Onutilisedeuxétats:quandontrouveunsymbole"e"après uneséquencedechiffresonstockelavaleurentièretrouvée dansunevariable,puisonvadansundeuxièmeétatoùonva lireexposant.

Nouvelleversionde `arith.flex` I

```
%%

%public
%class Lexer
%typeToken
%unicode

%{
    private Tokentoken(Symtype){
        returnnew Token(type);
    }
    private StringTokentoken(Symtype,Stringvalue){
        returnnew StringToken(type,value);
    }
    private IntTokentoken(Symtype, int value){
        returnnew IntToken(type,value);
    }
    int intbuff=0;
```


Nouvelleversionde `arith.flex` II

```
    private Stringchop(Strings){
        return (s.substring(0,s.length() -1));
    }
    privateint expo( int base, int ex){
      int result=base;
      for ( int i=1;i<=ex;i++){
        result=result *10;
      }
      return result;
}
%}

EspaceChar=[\n\r\f\t]
Ch =[0 -9]
Le =[a -zA-Z]

%stateEXPONENT

%%
```

Nouvelleversionde `arith.flex` III

```
<YYINITIAL>{
   {Ch}+ {                  return   token(Sym.INT,
                                          Integer.parseInt(yytext()));}
   {Le}({Le}|{Ch})   *   { return   token(Sym.IDENT,yytext());}
   "(" {                    return   token(Sym.PARG);}
   ")" {                    return   token(Sym.PARD);}
   "*" {                    return   token(Sym.MULT);}
   "+" {                    return   token(Sym.PLUS);}
   {EspaceChar}+ {}
   {Ch}+"e" {intbuff=Integer.parseInt(chop(yytext()));
                            yybegin(EXPONENT);}
}
<EXPONENT>{
   {Ch}+{yybegin(YYINITIAL);
          return  (token(Sym.INT,
                                 expo(intbuff,Integer.parseInt(yytext()))));}
}
```


Motsclefsd'unlangagedeprogrammation

Solutionnaïve:unerègleparmotclefs.

```
%%
%typeToken
 EspaceChar=[\n\r\f\t]
 Letter=[a              -zA-Z]
%%
 "begin" {          return   token(Sym.BEGIN)}
 "end" {            return   token(Sym.END)}
 "class" {          return   token(Sym.CLASS)}
 {Letter}+ {        return   token(Sym.IDENT,yytext());}
 {EspaceChar}{}
```


Attentionàl'ordredesrègles

Entrée: `begbeginbeginner`

- Premierappelà `yylex()` :seulementlaquatrièmerègle s'applique ⇒ tokenIDENT.
- Deuxièmeappelà `yylex()` :lesrègles(2)et(4)s'appliquent aumêmelexeme `begin`,c'estdonclapremièreparmicesdeux quigagne ⇒ tokenBEGIN.
- Troisièmeappelà `yylex()` :lesrègles(2)et(4)s'appliquent maisladernièrereconnaîtunlexemepluslong ⇒ token IDENT.

Maisregarderlatailledel'automateengendré!

Contrôlerlatailledel'automate

- Techniquesutiliséesparlegénérateurs:
  - Utiliserdesclassesdecaractèresaulieudanslareprésentation del'automate.
  - Minimiserl'automateengendréàpartirdesexpressions régulières.
- Optimisationdanslaspécification:Éviterdecréerune nouvelleclasselexicalepourchaquemotclef (Java:46motsclefs.)

Commentreconnaîtrelesmotsclefssanscatégoriesdédiées?

- EnJava(etpareildanslesautreslangagesde programmation):touslesmotsclefssontdesséquencesde lettresenminuscules.
- Mettreuneseulecatégoriepourlesidentificateurs.
- Dansl'actionassocié,oncherche(parex.dansunetablede hachage)silelexemeestunmotclefs,etcréeunjetonen fonction.

Lefichier `keys.flex` I

```
import java.util.HashMap;
class Keys extends HashMap<String,Sym>{
        public Keys(){
                super ();
                this .put("end",Sym.END);
                this .put("begin",Sym.BEGIN);
                this .put("class",Sym.CLASS);
        }
}

%%
%public
%typeToken
%class Lexer
%unicode
EspaceChar=[\n\r\f\t]
Letter =[a                -zA-Z]
```


Lefichier `keys.flex` II

```
%{
  private Keyskeys=      new Keys();
  private Tokentoken(SymtyKTokene1.09Tm0.01Tc[(K)10.(i)eturnn ee}privateTokentoken(Symsym,,eeunnnnrTKoyst=eys            et(unnnn);
```

AnalysedeDonnéesStructurées-Cours4

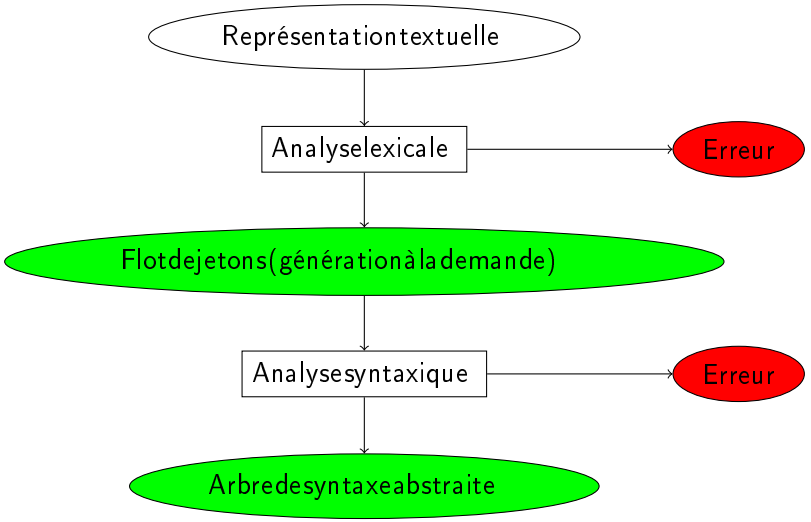
RalfTreinen

UniversitéParisDiderot
UFRInformatique
LaboratoirePreuves,ProgrammesetSystèmes

treinen@pps.univ-paris-diderot.fr

11février2015

Lerôledel'analysesyntaxique



Objectifdel'analysesyntaxique

- Deuxobjectifs:
  - Détecterdestextesd'entréequinesontpascorrectement formés(etdonnerdesindicationsutilessurlanaturede l'erreur).
  - Sil'entréeestcorrecte,construireunarbredesyntaxe abstraite.
- Focalisonsd'abordsurlepremierobjectif:distinguerdes textescorrectsdestextesincorrects.

Pourquoideuxétapesséparées?

- Onessayedefairetantd'analysequepossibledansl'analyse lexicale.
- Raison:l'exécutiond'unautomateesttrèsefficace(temps linéairedanslalongueurdutexted'entrée).
- Problème:l'expressivitédesautomatesestlimitée(voirles transparentssuivants).

Reconnaîtrelesexpressionsarithmétiques

- L'ensembledesexpressionsarithmétiquescorrectement parenthéséesn'estpasrégulier.
- Nousdémontreronsunesimplification: l'ensembledesmotsformésseulementdeparenthèses'['et']' quisontcorrectementimbriquées,n'estpasrégulier.
- Exemple d'unmotcorrectementimbriqué:

[[][[]]]

- Exemple d'unmotquin'estpascorrectementimbriqué:

[][[

Reconnaîtrel'imbricationcorrectedesparenthèses

Définitiondulangage P

P estlepluspetitlangagetelque

- $\in P$
- si $x; y \in P$ alors $xy \in P$
- si $x \in P$ alors $[x] \in P$

Théorème

P n'estpasrégulier.

Unlemme

Lemme:
Lelangage $L = \{[^n]^n \mid n \geq 1\}$ n'estpasrégulier.

Intuition

- L consisteentouslesmotsdelaforme

$$\underbrace{[\cdots[}_{n}\ \underbrace{]\cdots]}_{n}$$

pourun $n \geq 1$quelconque.

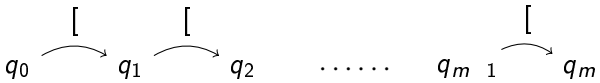
- Unautomatequiaccepte L devraitcompterles'[',etpuis comparercenombreaveclenombredes']'.
- Or,unautomatefininepeutpascompterunequantiténon bornée.

Dulemmeauthéorème

- Avantdedémontrerlelemme,réfléchissonspourquoilelemme impliquelethéorème:
- Supposonspourl'absurdeque P soitrégulier.
- Lelangage $L^0 = \{[^n]^m \mid n; m \geq 0\}$ estrégulier.
- $L = P \cap L^0$
- L'intersectiondedeuxlangagesréguliersestrégulier: contradiction!

Lapreuvedulemme

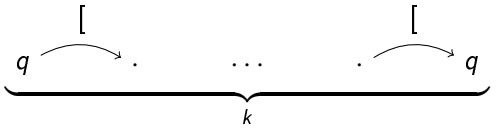
- Supposons,pourl'absurde,quel'automate $A = (Q; q_o; F; \Delta)$ accepte L.Soit m lenombred'élémentsde Q.
- A acceptedonclemot $[^m]^m$.
- Regardonslesactionsdel'automatequandillit m foisle symbole'[':



- Çafait $m + 1$étatsdanscetteséquence,maisiln'yaque m étatsdifférents.

Lapreuvedulemme

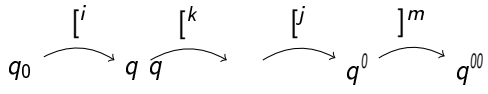
- Doncilyaunétat,disant q quiparaitdeuxfoisdanscette séquence:



- Soit k lalongueurdecetteséquence.Nousavonsdonc $\Delta\ (q; [^k) = q$.

Lapreuvedulemme

- Onadoncpourlemot $[^m]^m$:



avec $m = i + k + j$.

- Donc,onaaussi:

- Donc: $[^m\ \ ^k]^m \in \mathcal{L}(A)$,cequiestabsurde!

Lerésultatgénéral

Lemmedel'étoile(angl.: *pumpinglemma*)

Pourtoutlangageréguliер L ilexisteunentier m telquetoutmot $w \in L$ delongueur $|w| \geq m$ aunedécomposition $w = xyz$ telque

- $0 < |y|$
- $|xy| \leq m$
- $xy^n z \in$ pourtout $n \geq 0$

Démonstration

Commesurl'exempledestransparentsprécédents.

Définitiondesgrammairesalgébriques

Une *grammairealgébrique* (où: *grammairehorscontexte*)estun tuple $G = (V_T; V_N; S; R)$ où

- V_T estunensemblefinidesymbolesdits *terminaux*;
- V_N estunensemblefinidesymboles *non-terminaux*,où $V_N \cap V_T = \emptyset$;
- $S \in V_N$ estl' *axiome*;
- R estunensemblefinide *règles* delaforme $N \to u$,où $N \in V_N$,et $u \in (V_N \cup V_T)$.

Exempled'unegrammairealgébrique

$G_1 = (V_T; V_N; S; R)$ où

- $V_T = \{0; 1; +; *\}$
- $V_N = \{E; I\}$
- $S = E$
- R consisteenlesrèglessuivantes:

$$\begin{aligned} E &\to E + E \\ E &\to E * E \\ E &\to I \\ I &\to 0 \\ I &\to 1 \end{aligned}$$

Dérivation

Réécriture

Étantdonnéeunegrammaire $G = (V_T; V_N; S; R)$,onnote $V = V_T \cup V_N$.
Lemot $u \in V$ se *réécrit* enlemot $v \in V$ dans G,noté $u \to v$,si

1. $u = w_1 N w_2$,où $w_1 \in V$, $N \in V_N$ et $w_2 \in V$;
2. $N \to w$ estunerèglede R;
3. $v = w_1 w w_2$.

Dérivation

Lemot $v \in V$ *dérive* dumot $u \in V$,danslagrammaire G,noté $u \to v$,s'ilexisteunesuitefinie $w_0; w_1; \ldots; w_n$ demotsde V telleque $w_0 = u$, $w_i \to w_{i+1}$ pourtout $i \in [0; n-1]$,et $w_n = v$.

Exemple

- Pourlagrammaire G donnéeaudessusonalesréécritures suivantes:
  (Nousmettonsen rouge lenon-terminalquiestréécrit.)
  - $E \to E + E$
  - $E + E \to E + E + E$
  - $E + E + E \to E + E + E * E$
  - $E * E + E * E \to E * E + I * E$

Exemple:Dérivationde 1+0*1 dans G_1

$$
\begin{aligned}
E &\to E + E \\
&\to E + E * E \\
&\to I + E * E \\
&\to I + E * I \\
&\to I + E * 1 \\
&\to 1 + E * 1 \\
&\to 1 + I * 1 \\
&\to 1 + 0 * 1
\end{aligned}
$$

Langageengendré

Définition

Soit $G = (V_T; V_N; S; R)$ unegrammairealgébrique.Le *langage engendré* par G est

$$\mathcal{L}(G) = \{w \in V_T \mid S \to w\}$$

Unlangageest *algébrique* s'ilestengendréparunegrammaire algébrique.

Exemple

Pourlagrammaire G_1 onobtientque $\mathcal{L}(G_1)$ estl'ensembledes expressionsarithmétiquesforméesàl'aidedesopérateurs+et*,et desconstantes0et1,etsansparenthèses.

Relationaveclesla ngagesrégulières

- Toutlangagerégulierestalgébrique.
- Parcontre,ilyadeslangagesalgébriquesquinesontpas réguliers: $G_P = (\{[;]\}; \{E\}; E; R)$ où R consisteen

$$
\begin{aligned}
E &\to \\
E &\to EE \\
E &\to [\,E\,]
\end{aligned}
$$

Onaque $\mathcal{L}(G_P) = P$,lelangagedesséquencesdeparenthèses correctementimbriquées.

Notation:alternatives

- Unegrammairepeutavoirplusieursrèglesaveclemêmecôté gauche:

$$
\begin{aligned}
E &\to E + E \\
E &\to E * E
\end{aligned}
$$

- Nouspermettonsdanslasuitedanscecasd'écrireuneseule règle,avecplusieurs *alternatives* surlecôtédroite:

$$E \to E + E \mid E * E$$

Dérivationgauche

Soitunegrammaire $G = (V_T; V_N; S; R)$;onnote $V = V_T \cup V_N$.

Réécritureàgauche

Lemot $u \in V$ se *réécritàgauche* enlemot $v \in V$ dans G,noté $u \to_g v$,si

1. $u = w_1 N w_2$,où $w_1 \in V_T$, $N \in V_N$ et $w_2 \in V$;
2. $N \to w$ estunerèglede R ;
3. $v = w_1 w w_2$.

Dérivationgauche

Une *dérivationgauche* estunesuitefinie $w_0; w_1; \ldots; w_n$ demotsde V telleque $w_i \to_g w_{i+1}$ pourtout $i \in [0; n-1]$.

Exemple:Dérivationgauchede 1+0*1 dans G_1

$$
\begin{aligned}
E &\to E * E \\
&\to E + E * E \\
&\to I + E * E \\
&\to 1 + E * E \\
&\to 1 + I * E \\
&\to 1 + 0 * E \\
&\to 1 + 0 * I \\
&\to 1 + 0 * 1
\end{aligned}
$$

Arbrededérivation

Définition

Soit $G = (V_T; V_N; S; R)$ unegrammaire.Un *arbrededérivation* de G estunarbretelque

- lesnœudsinternessontétiquetéspardessymbolesde V_N ;
- lesfeuillessontétiquetéespardessymbolesde $V_T \cup V_n$;
- silesfilsprisdegaucheàdroited'unnœudinterneétiqueté parlenon-terminal N sontétiquetésparlessymboles respectifs $_1; \ldots; _n$,alors $(N \to _1 \cdots _n) \in R$.

Définition

Unarbrededérivationdontlaracineestétiquetéeparl'axiomede lagrammaireetdontlemotdesfeuilles u appartientà V_T est appelé *arbrededérivationdeu* .

Exemple:Unarbrededérivationde 1+0*1 dans G

D'unedérivationgaucheà l'arbrededérivation

- Laracinedel'arbreestétiquetéeparl'axiomedelagrammaire (quiestaussilepremierélémentdeladérivationgauche).
- Onparcourtladérivationdegaucheàdroite,encomplétant l'arbre.
- Àlaréécrituredupremiernon-terminaldansunmot correspondl'extensiondelafeuilledel'arbrededérivationen constructionlaplusgauchequiestétiquetéeparun non-terminal.
- (Voirl'exempledonnéautableau)

AnalysedeDonnéesStructurées-Cours4
└Grammaires

Del'arbrededérivationàladérivationgauche

- Lepremierélémentdeladérivationgaucheestl'axiomedela grammaire(quiestaussil'étiquettedelaracinedel'arbre).
- Onfaitunparcourspréfixedel'arbre,encomplétantla dérivationgauche.
- Auxfilsd'unnœudinternecorrespondlaréécrituredupremier non-terminaldansunélémentdeladérivation.
- (Voirl'exempledonnéautableau)

AnalysedeDonnéesStructurées-Cours4
└Grammaires

ArbresdedérivationetDérivationgauches

- Équivalenceentredérivationsgauchesetarbresdedérivation:
  - Pourchaquearbrededérivationilyuneuniquedérivation gauche.
  - Pourchaquedérivationgaucheilyaununiquearbrede dérivation.
- Dansunpremiertemps,onpeutimaginerl'arbrededérivation commerésultatdel'analysesyntaxique.
- Ilnousfautdoncquechaquemotdulangagepossèdeunarbre dedérivationunique.

AnalysedeDonnéesStructurées-Cours4
└Grammaires

Grammairesambiguës

Définition

Unegrammaire G estnon-ambiguëquandtout $w \in \mathcal{L}(G)$ aunseul arbrededérivation.

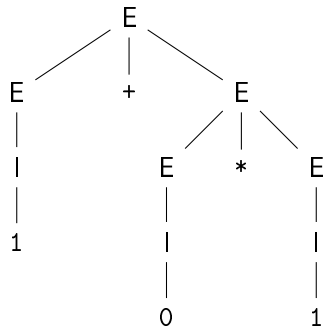
Définitionéquivalente

Unegrammaire G estnon-ambiguëquandtout $w \in \mathcal{L}(G)$ aune seuledérivationgauche.

Surl'exemple G_1

Lagrammaire G_1 estambiguë:lemot 1+0*1 adeuxarbresde dérivationdifférents!

Unautrearbrededérivationde 1+0*1 dans G_1



Unautreexemple

- Lagrammaire $G_2 = (\{\mathtt{i}; \mathtt{v}; +; *; (;)\}; \{\mathrm{E}\}; \mathrm{E}; R)$, où R est

$$\mathrm{E} \rightarrow \mathtt{i} \mid \mathtt{v} \mid (\mathrm{E} + \mathrm{E}) \mid (\mathrm{E} * \mathrm{E})$$

- Cettegrammairedécritlesexpressionsarithmétiques complètementparanthésées.
- Iciilyaunseulterminal i pourlesentiers,etunseulterminal v pourlesnomsdesvariablescarenimaginequ'ils'agitdes jetonsissusd'uneanalyselexicale.

G_2 estnon-ambiguë

- Pourledémontrernousavonsbesoindedeuxpetitslemmes.
- Utilitédeceslemmes:Imaginezqu'unmot w àdeux dérivationsgauchesquisontdelaforme

$$\mathrm{E} \rightarrow (\mathrm{E} + \mathrm{E}) \rightarrow w$$
$$\mathrm{E} \rightarrow (\mathrm{E} * \mathrm{E}) \rightarrow w$$

- Soit w_1 lemotdérivéduEgauchedelapremièreligne, w_2 le motdérivéduEgauchedeladeuxièmeligne.
- Forcement, $w_1 \neq w_2$.Donc, $w_1 <_{\text{pref}} w_2$ (oul'inverse).
- Leslemmesnouspermettentdedémontrerquec'est impossibled'avoir $w_1; w_2 \in \mathcal{L}(G_2)$ avec $w_1 <_{\text{pref}} w_2$.

Séparer $\mathcal{L}(G_2)$ etsespropresprefixes

Notation

- $|w|_($ estlenombredesymboles(danslemot w.
- $|w|_)$ estlenombredesymboles)danslemot w.

Lemme1

Pourtous $w \in \mathcal{L}(G_2) : |w|_(= |w|_)$.

Démonstration

Parinductionsurlalongueurdeladérivationlaplus courtede w (autableau).

Séparer $\mathcal{L}(G_2)$ etsespropresprefixes

Lemme2
Pourtous $w \in \mathcal{L}(G_2)$:pourtous $v <_{\text{pref}} w$ avec $v \neq$:
$|v|_(> |v|_)$.

Démonstration
Parinductionsurlalongueurdeladérivationlapluscourtede w
(autableau).

G_2 estnon-ambiguë

- Supposonspourl'absurdeque G_2 estambiguë.
- Soit w unmotdelongueurminimalequiadeuxdérivations gauches.
- Iln'estpaspossiblequ'unedesdeuxcommencesurE $\rightarrow$ i ou E $\rightarrow$ v.
- Donc,unecommencesurE $\rightarrow (E \circ E)$,etl'autresur E $\rightarrow (E \diamond E)$,avec $\circ;\diamond \in \{+;*\}$.
- Donc:

$$\begin{aligned} w &= (v_1 \circ v_2) \\ w &= (u_1 \diamond u_2) \end{aligned}$$

avecE $\rightarrow$ $v_1; v_2; u_1; u_2$,delongueurspluspetitesque w.
Chacundecesmotsadoncuneseuledérivationgauche!

G_2 estnon-ambiguë

- Soit $v_1 = u_1$,alorsforcement $\circ = \diamond$, $v_2 = u_2$,etlesdeux dérivationsgauchessontlesmêmes.Contradiction!
- Soit $v_1 <_{\text{pref}} u_1$.Onadonc $v_1 \in \mathcal{L}(G_2)$,et $v_1 <_{\text{pref}} u_1 \in \mathcal{L}(G_2)$:ContradictionauxLemmes1et2!
- Soit $u_1 <_{\text{pref}} v_1$.Onadonc $u_1 \in \mathcal{L}(G_2)$,et $u_1 <_{\text{pref}} v_1 \in \mathcal{L}(G_2)$:ContradictionauxLemmes1et2!

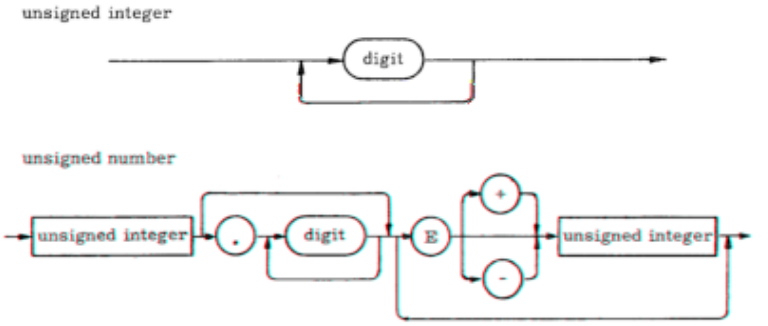
Backus-NaurForm

- Unenotationtrèsutiliséepourladocumentationdeslangages deprogrammationestlanotation *BNF* (Backus-NaurForm) quiestéquivalenteauxgrammairesalgébriques.
- Lesnon-terminauxsontécritentrechevrons: $\langle T \rangle$.
- Laflècheestremplacéepar ::=
- Lesalternativespourlemêmenon-terminalsontregroupées commeindiquéesau-dessus.
- Exemple:
<cond>::=if"("<condition>")""{"<code>"}"

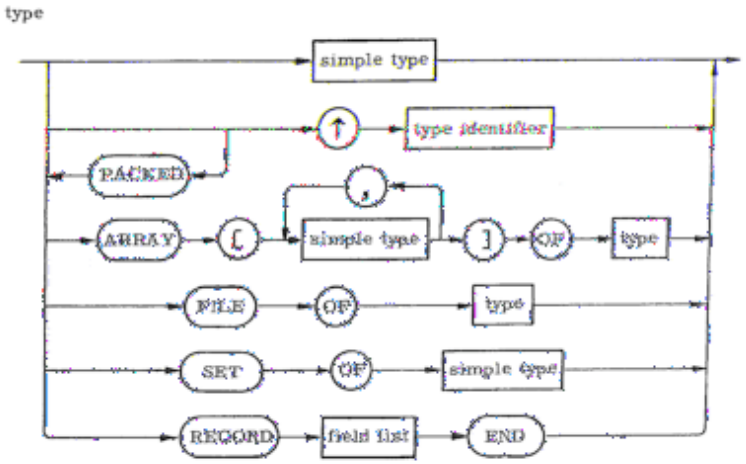
ExtendedBackus-NaurForm

- Laformeétenduepermetaussidesconstructionsdes expressionsrégulièresdanslagrammaire:opérateurs $*$ et $+$, etparenthèses.
- Unepartieentrecrochets[]estoptionnelle.
- Exemple: `<explist>::="("<exp>(","<exp>)*")"`
- C'estencoreéquivalentauxgrammairesalgébriques.

Diagrammedesyntaxe(non-récursif)



Diagrammedesyntaxe(récursif)



AnalysedeDonnéesStructurées-Cours5

RalfTreinen

UniversitéParisDiderot
UFRInformatique
LaboratoirePreuves,ProgrammesetSystèmes

treinen@pps.univ-paris-diderot.fr

18février2015

L'objectif

- L'analysesyntaxiqueadeuxobjectifs:
  - détectersiletexteluestcorrect(estdanslelangageengendré parlagrammaire);
  - lecaséchéant,construireunarbredesyntaxeabstraite.
- Pourl'instant,nouscontinuonsàétudierlepremierproblème: reconnaissancedestextesd'entréecorrectes.

I ▶L'analyse:

Déterminiser un automate fini

- À chaque moment de l'exécution d'un automate fini, sa configuration consiste en :
  - l'information où on est dans la lecture du mot d'entrée, et
  - l'état de l'automate (nombre fini).
- L'ensemble des configurations possibles dans lesquelles l'automate peut se trouver après lecture d'un mot donné est borné (par le nombre d'états de l'automate).
- Construction d'un automate déterministe de taille exponentielle.

Déterminiser un automate à pile ?

- La configuration d'un automate à pile est plus riche :
  - l'information où on est dans la lecture du mot d'entrée,
  - l'état de l'automate,
  - et le *contenu de la pile* .
- La taille de la pile est *non-bornée*.
- Le nombre de configurations possibles dans lesquelles l'automate peut se trouver après lecture d'un mot n'est plus borné !

Plusieurs techniques pour l'analyse syntaxique

- Analyse descendante : construction (virtuelle) de l'arbre de dérivation, à partir de l'axiome aux feuilles.
  Ordre de construction : parcours préfixe de l'arbre.
  C'est l'approche présentée dans ce cours.
- Analyse ascendante : construction (virtuelle) d'un arbre de dérivation à partir des feuilles jusqu'à l'axiome. Plus complexes à maîtriser, nécessite des connaissances des automates à pile.
  Voir le cours de Compilation au M1.

Construction (virtuelle) d'un arbre de dérivation

- Dans la construction d'un arbre de dérivation (ou, d'une dérivation), il y a à chaque moment deux choix à faire :
  - du non-terminal qu'on va remplacer à l'aide d'une règle de la grammaire,
  - une fois le non-terminal choisi, de la règle parmi ceux qui ont ce non-terminal sur le côté gauche.
- Nous avons vu la semaine dernière que le premier choix n'est pas essentiel : on peut imposer une stratégie comment choisir le non-terminal à remplacer (par ex., celui qui est le plus à gauche).

Explorationcomplètedel'espacederecherche?

- Unefaçonderéaliseruneanalysesyntaxiqueestmaintenant d'essayersimplementtouteslespossibilitésdechoisirdes règles.
- Celadonnerlieuàunalgorithmenon-déterministe:
  - soitpar *retour-en-arrière* (angl.:backtracking)
  - soitpar *programmationdynamique*
- Approchecomplète:onestsûrdetrouverunarbrede dérivationsilemotestdanslelangage ☺
- Problème:efficacité ☹
- Onchercheàdessolutionsefficaces,éventuellementen imposantdesrestrictionsauxgrammairesqu'onpeuttraiter.

Commentobtenirunesolutionefficace?

- Ilfautmaîtriserlechoixdelarègledelagrammairepar laquelleonvaremplacerunnon-terminal.
- Onnepeutpasdemanderqu'ilyaituneseulerèglepar non-terminal(cardanscecaslagrammaireestcomplètement triviale).
- Surquoibaserlechoixdelarègle?
- Surlasuitedumotpourlequeloncherche construirel'arbre dedérivation!

Exemple

- Grammaire $G = (V_T, V_N, S, R)$ où
- $V_T = \{\texttt{i}, +, [,]\}$
- $V_N = \{\text{S}\}$
- $S = \text{S}$
- R consisteenlesrèglessuivantes:

$$\text{S} \rightarrow i \quad (1)$$
$$\text{S} \rightarrow [\text{S+S}] \quad (2)$$

- $\mathcal{L}(G)$:expressionscomplètementparenthésées,construites aveclaconstante `i` etl'opérateurbinaire `+`.

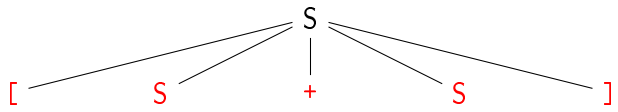
Constructiond'unarbrededérivation(1)

S

[	[	i	+	i	]	+	[	i	+	i	]	]

Choisirrègle(2):c'estlaseulequipeutproduireàpartirdeSun motquicommencesur `[`.

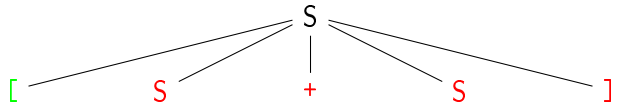
Constructiond'unarbrededérivation(2)



[	[	i	+	i	]	+	[	i	+	i	]	]

Lepremiernon-terminaldumotdesfeuillesest [.

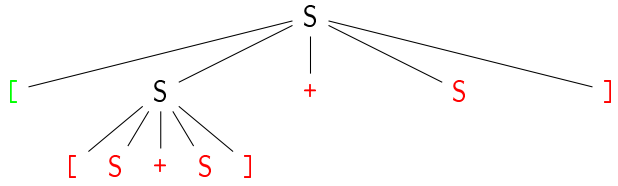
Constructiond'unarbrededérivation(4)



[	[	i	+	i	]	+	[	i	+	i	]	]

Choisirrègle(2):c'estlaseulequipeutproduireàpartirdeSun motquicommencesur [.

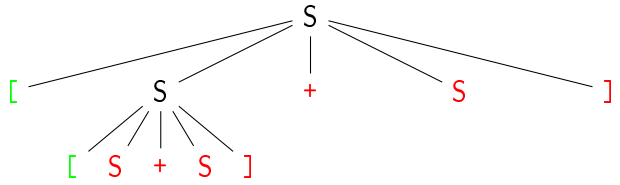
Constructiond'unarbrededérivation(5)



[	[	i	+	i	]	+	[	i	+	i	]	]

Lesuivantnon-terminaldumotdesfeuillesest [.

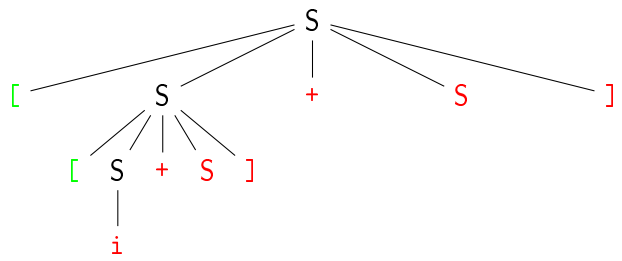
Constructiond'unarbrededérivation(6)



[	[	i	+	i	]	+	[	i	+	i	]	]

Choisirrègle(1):c'estlaseulequipeutproduireàpartirdeSun motquicommencesur i.

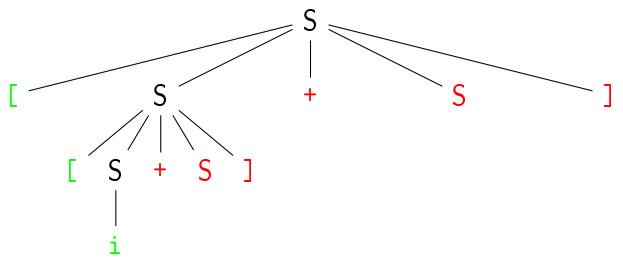
Constructiond'unarbrededérivation(7)



[	[	i	+	i	]	+	[	i	+	i	]	]

Lesuivantnon-terminaldumotdesfeuillesest `i`.

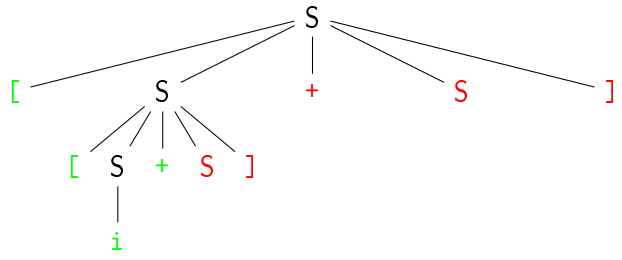
Constructiond'unarbrededérivation(8)



[	[	i	+	i	]	+	[	i	+	i	]	]

Lesuivantnon-terminaldumotdesfeuillesest `+`.

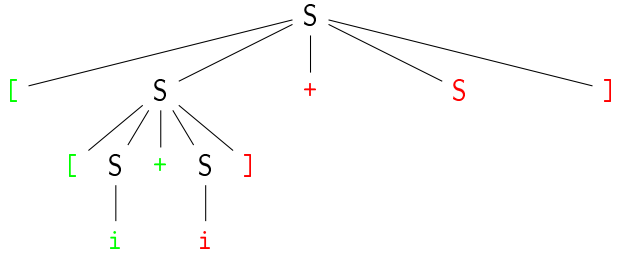
Constructiond'unarbrededérivation(9)



[	[	i	+	i	]	+	[	i	+	i	]	]

Choisirrègle(1):c'estlaseulequipeutproduireàpartirdeSun motquicommencesur `i`.

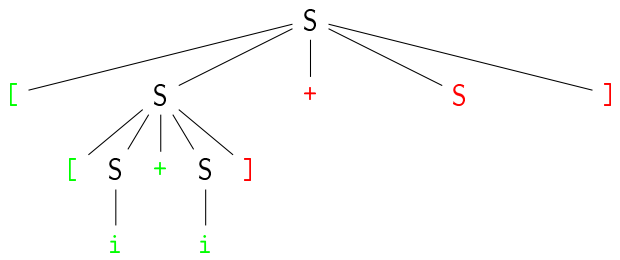
Constructiond'unarbrededérivation(10)



[	[	i	+	i	]	+	[	i	+	i	]	]

Lesuivantnon-terminaldumotdesfeuillesest

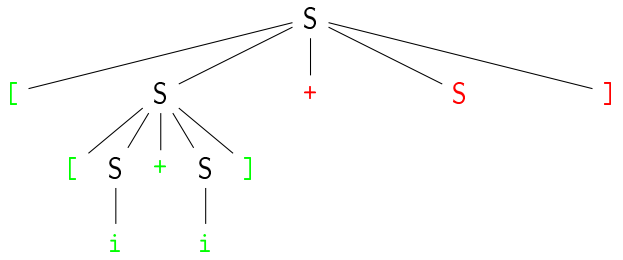
Constructiond'unarbrededérivation(11)



[	[	i	+	i	]	+	[	i	+	i	]	]

Lesuivantnon-terminaldumotdesfeuillesest].

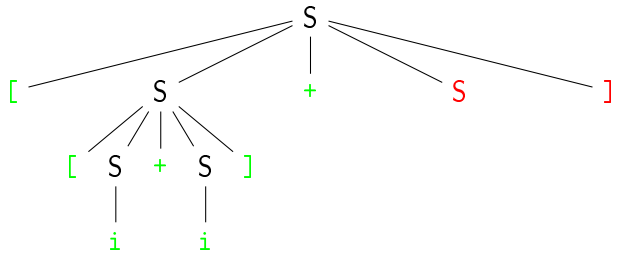
Constructiond'unarbrededérivation(12)



[	[	i	+	i	]	+	[	i	+	i	]	]

Seulementlarègle(2)peutproduireàpartirdeSunmotqui commencesur +.

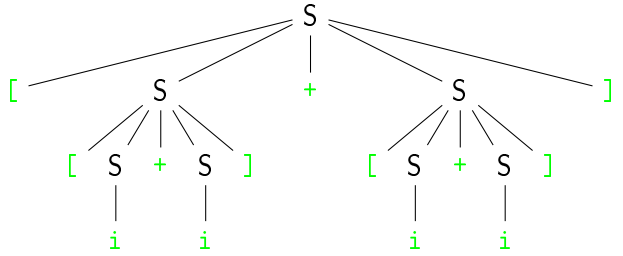
Constructiond'unarbrededérivation(13)



[	[	i	+	i	]	+	[	i	+	i	]	]

Choisirrègle(2):c'estlaseulequipeutproduireàpartirdeSun motquicommencesur [.

etc.etc.



[	[	i	+	i	]	+	[	i	+	i	]	]

Constructionterminée!

Cequ'onavusurl'exemple:

- Ilyadeuxtypesd'actions:
  - consommerenparallèleunnon-terminaldupréfixedumotdes feuillesdéjàconstruit,etlemêmesymboledel'entrée;
  - ajouterdesfilsàunefeuilledel'arbrededérivationpartiel.
- Pourchoisirlarègledelagrammaire,onregardeenavantquel estlesymbolesuivantdel'entréequenousaurionsà consommer(lookahead).

GrammairesLL(1)

Enfait,l'algorithmequenousavonsvusurl'exempleappartientà laclasse *LL(1)* :

- lepremier *L* indiquequ'onparcourtl'entréedelagauche (angl.:left)àladroite;
- ledeuxième *L* indiquequ'onconstruitunedérivationgauche (angl.:left),c.-à-d.unarbrededérivationdansunordre préfixe;
- lenombre *1* indiquequenousutilisonslaconnaissancede1 caractèredanslapartiedel'entréequiresteàconsommer, pourdéterminerlarègleàappliquer(lookahead=1).

Notation: $w : k$

Définition

Soit $w \in \Sigma^*$ unmot,et $k \in \mathbb{N}$.Ondéfinit

- si $|w| \leq k$ alors $w : k = w$
- si $|w| > k$ alors $w : k = x$ telque $w = xy$ et $|x| = k$

Explication

- $w : k$ estlepréfixedelongueur k dumot w,oulemot w entiersi w estpluscourtque k.
- $abcdefg : 3 = abc$
- $abcd : 7 = abcd$

DéfinitionLL(k)

Définition

Soit $G = (V_T, V_N, S, R)$ unegrammairealgébrique, $k \in \mathbb{N}$. G est dite *LL(k)* ssi

- S'ilexistedeuxdérivationsgauches

$$S \rightarrow^* uY\alpha \rightarrow u\beta\alpha \rightarrow^* ux$$
$$S \rightarrow^* uY\alpha \rightarrow u\gamma\alpha \rightarrow^* uy$$

où $Y \in V_N$, $u, x, y \in V_T^*$, $\alpha, \beta, \gamma \in (V_N \cup V_T)^*$,avec $x : k = y : k$

- alors $\beta = \gamma$.

Explicationdeladéfinitionde LL(k)

- On a déjà consommé le mot de terminaux u.
- Le non-terminal le plus à gauche à réécrire est maintenant Y.
- Dans les deux cas considérés, le mot d'entrée continu une fois par le mot x, l'autre fois par le mot y.
- En regardant les k premiers caractères de la suite du mot d'entrée, on peut maintenant décider comment réécrire le non-terminal y.

AnalysedeDonnéesStructurées-Cours5
└GrammairesLL(k)

Unpremiercritèresimplepourêtre LL(1)

Lemme

Si pour tout non-terminal, les côtés droites de toutes les règles pour ce non-terminal commencent par des terminaux différents, alors la grammaire est LL(1).

Exemple

La grammaire de l'exemple précédent:

$$\begin{aligned} S &\to i \\ S &\to [S+S] \end{aligned}$$

satisfait le critère, et est donc LL(1).

AnalysedeDonnéesStructurées-Cours5
└GrammairesLL(k)

Unmeilleurcritèrepourêtre LL(1)?

Problème

Le critère du transparent précédent est un peu trop restrictif car il ne permet pas des règles où le côté droite commence par un non-terminal:

$$\begin{aligned} A &\to BA \mid B \quad (3) \\ B &\to \ldots \quad (4) \end{aligned}$$

AnalysedeDonnéesStructurées-Cours5
└GrammairesLL(k)

Lafonction FIRST_k

Définition

Soit $G = (V_T, V_N, S, R)$ une grammaire, et $k \in \mathbb{N}$. Nous définissons une fonction

$$\mathrm{FIRST}_k : (V_T \cup V_N)^* \to 2^{V_T^*}$$

par

$$\mathrm{FIRST}_k(\alpha) = \{w : k \mid w \in V_T^*, \alpha \to^* w\}$$

Explication

$\mathrm{FIRST}_k(\alpha)$ est l'ensemble des préfixes de longueur k des mots terminaux qu'on peut obtenir à partir de α.

UnmeilleurcritèrepourêtreLL(1)

Lemme

Soit $G = (V_T, V_N, S, R)$ unegrammaire *sansproductionsdela formeN* $\rightarrow \epsilon$. G estLL(1)siestseulementsipourtoutesrègles différentes:

$$\begin{array}{rcl} \mathrm{N} & \rightarrow & \alpha \\ \mathrm{N} & \rightarrow & \beta \end{array}$$

onaque $\mathrm{FIRST}_1(\alpha) \cap \mathrm{FIRST}_1(\beta) = \emptyset$.

Exemple

Toujourssurlemêmeexemple:

$$\begin{array}{rcl} \mathrm{FIRST}_1(\mathtt{i}) & = & \{\mathtt{i}\} \\ \mathrm{FIRST}_1(\mathtt{[S+S]}) & = & \{\mathtt{[}\} \end{array}$$

Calculde FIRST_1

- Grammaire $G = (V_T, V_N, S, R)$. *Hypothèse:aucune productionN* $\rightarrow \epsilon$.
- Oncalcule $Fi(\alpha)$ pourtoutcôtédroitede R.
- $Fi(a\alpha) = \{a\}$ pourtout $a \in V_T$
  $Fi(N\alpha) = \emptyset$ pourtout $N \in V_N$
- Tantqu'ilexisteunerègle $N \rightarrow \beta$ telleque $Fi(\beta) \not\subseteq Fi(N\alpha)$:

$$Fi(N\alpha) = Fi(N\alpha) \cup Fi(\beta)$$

- Pourtout α : $\mathrm{FIRST}_1(\alpha)$ estlavaleurfinalede $Fi(\alpha)$.

Exemple

- Grammaire $G = (\{\ , (,), +\}, \{\mathrm{F}, \mathrm{S}\}, \mathrm{S}, R)$ où R est

$$\begin{array}{rcl} \mathrm{F} & \rightarrow & \\ \mathrm{S} & \rightarrow & (\mathrm{F+S}) \\ \mathrm{S} & \rightarrow & \mathrm{F} \end{array}$$

- Initialisation:

$$\begin{array}{rcl} Fi(\ ) & = & \{\ \} \\ Fi((\mathrm{F+S}) & = & \{(\} \\ Fi(\mathrm{F}) & = & \emptyset \end{array}$$

- Complétion:Onaunerègle$\mathrm{F}$ $\rightarrow$ a,mais $Fi(\ )\neg \subseteq Fi(\mathrm{F})$.
- Onaugmente: $Fi(\mathrm{F}) = \{\ \}$

Exemple(2)

- Grammaire $G = (\{\ , (,), +\}, \{\mathrm{F}, \mathrm{S}\}, \mathrm{S}, R)$ où R est

$$\begin{array}{rcl} \mathrm{F} & \rightarrow & \\ \mathrm{S} & \rightarrow & (\mathrm{F+S}) \\ \mathrm{S} & \rightarrow & \mathrm{F} \end{array}$$

- Onobtientdonc:

$$\begin{array}{rcl} \mathrm{FIRST}_1(\ ) & = & \{\ \} \\ \mathrm{FIRST}_1((\mathrm{F+S})) & = & \{(\} \\ \mathrm{FIRST}_1(\mathrm{F}) & = & \{\} \end{array}$$

Structureducode

- Nousavonsbesoinderegarderlesymbolesuivantdansleflot d'entréesansdeleconsommer:classe `LookAhe d1Re der`.
- L'analyseursyntaxiquecontientuneméthode `term(ch rc)` quiconsommeunsymbole c duflotd'entrée.
- L'analyseursyntaxiquecontientuneméthode `nonterm_N()` pourchaquenon-terminal N.Cetteméthodeconsommeun motduflotd'entréequiestengendrépar N. L'implémentationdecesméthodesutilise FIRST_1 pour déterminerlarègledelagrammaireàappliquer.

Fichier LookAhead1Reader.java I

```
import java.io.*;

class LookAhead1Reader extends PushbackReader{
    /* Readerclasswithalookaheadofonecharacter */

    public LookAhead1Reader(Readerr){
        super(r,1);
    }

    publicboolean check(char c)
        throws IOException{
        /* checkwhethercisthefirstcharacter */
        int lastread=0;
        lastread=this.read();
        this.unread(lastread);
        return (lastread==c);
    }
```


Fichier LookAhead1Reader.java II

```
    publicvoid eat(char expected)
        throws ReadException,IOException{
        /* consumescfromthestream,exception */
        /* whenthecontentsdoesnotstartonc. */
        char found=(char)this.read();
        if (found!=expected){
            thrownew ReadException(expected,found);
        }
    }
}
```


Fichier Parser.java I

```
import java.io.*;

/* simpleLL(1)parserforthegrammar: */
/* F ->aS ->FS ->(F+S) */
class Parser{

    protected LookAhead1Readerreader;

    public Parser(LookAhead1Readerr){
        reader=r;
    }
```

Fichier Parser.java II

```
public void term(char c) throws IOException,ReadException{
    /* consumethecharacterc */
    reader.eat(c);
}

public void nonterm_S()
    throws IOException,IllegalArgumentException,
           ReadException,ParserException{
    /* parseawordgeneratedfromnonterminalS */
    if (reader.check('a')){
        this.nonterm_F();
    } elseif (reader.check('(')){
```


Fichier Parser.java III

```
            this.term('(');
            this.nonterm_F();
            this.term('+');
            this.nonterm_S();
            this.term(')');
        } else {
            thrownew ParserException("cannot␣reduce␣S");
        }
    }

    public void nonterm_F() throws IOException,ReadException{
        /* parseawordgeneratedfromnonterminalF */
```


Fichier Parser.java IV

```
            this.term('a');
        }
    }
```


Fichier Test.java

```
import java.io.*;

class Test{

    public static void main(String[]args) throws Exception{
        File input= new File(args[0]);
        Reader reader= new FileReader(input);
        LookAhead1Reader r= new LookAhead1Reader(reader);
        Parser p= new Parser(r);
        p.nonterm_S();
    }
}
```

AnalysedeDonnéesStructurées-Cours6

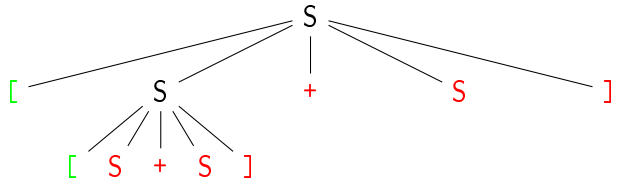
RalfTreinen

UniversitéParisDiderot
UFRInformatique
LaboratoirePreuves,ProgrammesetSystèmes

treinen@pps.univ-paris-diderot.fr

11mars2015

Constructiond'unarbredededérivation



Choisirrègle(1):c'estlaseulequipeutproduireàpartirdeSun motquicommencesur `i`.

GrammairesLL(1)

- Intuitionderrièrelesgrammaires LL(1):danslaconstruction d'unedérivationgauche,lesymbolesuivantdel'entréenous indiquequellerègledelagrammaireappliqueraunon-terminal leplusàgauchedel'arbredededérivation.
- Conséquence:toutegrammaireLL(1)(oumêmeLL(k))est non-ambiguë(oubliédepréciserauderniercours).
- Uncritèretrèssimple:Sitouslescôtésdroitesdela grammairepourle *même* non-terminalcommencentsurdes terminauxdifférents,alorslagrammaireestLL(1).

UnecaractérisationdesgrammairesLL(k)

Théorème
Unegrammaire $G = (V_T; V_N; S; R)$ estLL(k)ssi

- si $A \rightarrow$ et $A \rightarrow$ sontdeuxrèglesdifférentes
- et $S \rightarrow^* wA$ unedérivationgauche, $w \in V_T^*$
- alors $\mathrm{FIRST}_k(\quad) \cap \mathrm{FIRST}_k(\quad) = \emptyset$

- Preuveomise(conséquenceimmédiatedeladéfinition)
- Attention,cecritèregénéralprendencomptele"contexte" danslequelonpeutobtenir A.

UnmeilleurcritèrepourêtreLL(1)

- Définitionde $FIRST_1(\ )$:l'ensembledessymbolesavec lesquellesunmotterminaldérivéàpartirde peut commencer(plus danslecasoù $\rightarrow^*$).
- Vuauderniercours:calculde $FIRST_1$ danslecasoùaucun côtédroiteest .
- Meilleurcritère:Sitouslescôtésdroitesdelagrammairepour le *même* non-terminalontdesensembles $FIRST_1$ disjointes, alorslagrammaireestLL(1).

AnalysedeDonnéesStructurées-Cours6
└RappeletCompléments:LL(1)etLL(k)dansdescassimples

Exemplevuauderniercours

- Grammaire $G = (\{\ ;(;);+\};\{\mathrm{F};\mathrm{S}\};\mathrm{S};R)$ où R est

$$\begin{aligned} \mathrm{F} &\rightarrow \\ \mathrm{S} &\rightarrow (\mathrm{F}+\mathrm{S}) \\ \mathrm{S} &\rightarrow \mathrm{F} \end{aligned}$$

- Lepremièrecritèresimplenes'appliquepas.
- Onobtientpourlescôtésdroitesdesrègles:

$$\begin{aligned} \mathrm{FIRST}_1(\ ) &= \{\ \} \\ \mathrm{FIRST}_1((\mathrm{F}+\mathrm{S})) &= \{(\} \\ \mathrm{FIRST}_1(\mathrm{F}) &= \{\ \} \end{aligned}$$

AnalysedeDonnéesStructurées-Cours6
└RappeletCompléments:LL(1)etLL(k)dansdescassimples

Reconnaîtrelafindel'entrée

- Leprogrammevuauderniercoursaundéfaut:ilaccepte aussidesexpressionscorrectes,avecuntextequelconque ajoutéàlafin: `( +( + ))%$#@`
- Solution:
  - l'analyselexicaleenvoieunjetonquisignalelafindel'entrée (parexemple,EOF)
  - remplacerl'axiomeparS ′,avecunerègle

$$\mathrm{S}' \rightarrow \mathrm{S}\ \texttt{EOF}$$

  oùSestl'ancienaxiomedelagrammaire.

AnalysedeDonnéesStructurées-Cours6
└RappeletCompléments:LL(1)etLL(k)dansdescassimples

Lemêmeexempleavecreconnaissancedelafin

- Grammaire $G = (\{\ ;(;);+;\texttt{EOF}\};\{\mathrm{F};\mathrm{S};\mathrm{S}'\};\mathrm{S}';R)$ où R est

$$\begin{aligned} \mathrm{F} &\rightarrow \\ \mathrm{S} &\rightarrow (\mathrm{F}+\mathrm{S}) \\ \mathrm{S} &\rightarrow \mathrm{F} \\ \mathrm{S}' &\rightarrow \mathrm{S}\ \texttt{EOF} \end{aligned}$$

- Onobtientpourlescôtésdroitesdesrègles:

$$\begin{aligned} \mathrm{FIRST}_1(\ ) &= \{\ \} \\ \mathrm{FIRST}_1((\mathrm{F}+\mathrm{S})) &= \{(\} \\ \mathrm{FIRST}_1(\mathrm{F}) &= \{\ \} \\ \mathrm{FIRST}_1(\mathrm{S}\ \texttt{EOF}) &= \{\ ;(\} \end{aligned}$$

Unexempleunpeuplusavancé

- Unegrammairepourlesexpressionsarithmétiques partiellementparenthésées.
- Terminaux: {i; +; *; (;); EOF}
- Règles:

$$\begin{aligned} S &\to E\ \texttt{EOF} \\ E &\to E\texttt{+}T \mid T \\ T &\to T\texttt{*}F \mid F \\ F &\to \texttt{(}E\texttt{)} \mid \texttt{i} \end{aligned}$$

- Axiome:S

L'exemplesexpressionspartiellementparenthésées

- Cettegrammaireestnon-ambiguë ☺
- Intuition:Les" +"peuventêtreproduitesseulementàpartir duE.Toutmotengendréparestprotégépardes parenthèses (et).
- (Ilyaaussiunepreuveformellemaisjevousenfaitgrâce.)
- Cettegrammaire,est-elleaussiLL(1)?OuaumoinsLL(k) pourquelque $k \in \mathbb{N}$?
- Ellen'estpasLL(k),pouraucun k ! ☹

Uncritèrepournepas *pas* êtreLL(k)

Définition

Unegrammaire $G = (V_T; V_N; S; R)$ est *récursiveàgauche* s'ilya unnon-terminal $N \in V_N$ telque $N \to^+ N$ pourun $\in (V_T \cup V_N)^*$.

- $\to^+$:dérivationenaumoinsuneétape.
- Exemple:notregrammairepourlesexpressionspartiellement parenthésées,carE $\to$ E+T

Lemme

Silagrammaire G estrécursiveàgauche,alors G n'estpas *pas* LL(k),pouraucun $k \in \mathbb{N}$.

Preuve

- Supposonspourl'absurdeque G estLL(k)etrécursiveà gauche.
- Ilyadoncunedérivationgauche $S \to^* wX$ (sinon X n'est pasatteignable).
- Ilyaaussiunerègle $X \to X$ (hypothèsesimplificatrice),et unerègledifférente $X \to$ (sinon X estnon-productif).
- Ilexistedoncunedérivationgauche

$$S \to^* wX \to^* wX^{\,k}$$

- Parlethéorèmedudébutducours:

$$\mathrm{FIRST}_k(X^{\,k+1}) \cap \mathrm{FIRST}_k(\quad) = \emptyset$$

Preuve(2)

- Ona:

$$\mathrm{FIRST}_k(X^{k+1}) \cap \mathrm{FIRST}_k(^{k}) = \emptyset$$

- Donc,grâceàlarègle $X \rightarrow$:

$$\mathrm{FIRST}_k(^{k+1}) \cap \mathrm{FIRST}_k(^{k}) = \emptyset$$

- Contradiction(deuxcas: $\rightarrow^*$ oupas).

Quoifaire?

- Onpeuttransformerlagrammaireenunegrammaire équivalente(quidéfinitlemêmelangage),etquiestLL(1).
- C'esttoujourspossible,maisilyadeuxinconvénients:
  - lagrammairerésultantepeuxêtreplusgrande;
  - lastructuredel'arbrededérivationpeutchanger.
- Ilyauntroisièmeinconvénient:latransformationpeut introduiredesrègles $N \rightarrow$,ilfautdoncadapterlatechnique àcecas.

Lagrammairetransformée

- Unegrammairepourlesexpressionsarithmétiques partiellementparenthésées.
- Terminaux: {i;+;*;(;);EOF}
- Règles:

$$\begin{aligned} S &\rightarrow E\ \mathtt{EOF} \\ E &\rightarrow TE' \\ E' &\rightarrow \ \mid +E \\ T &\rightarrow FT' \\ T' &\rightarrow \ \mid *T \\ F &\rightarrow (E) \mid \mathtt{i} \end{aligned}$$

- Axiome:S

Explicationdelatransformation

- Lesdeuxrèglesoriginalespourlenon-terminalE:

$$\begin{aligned} E &\rightarrow T \\ E &\rightarrow E{+}T \end{aligned}$$

- Danslagrammaired'origine,lenon-terminalEengendreune séquencenon-videdenon-terminauxT,séparéspardes+.
- Danslanouvellegrammaire,lenon-terminalE ′ engendrela suitedecetteséquenceaprèsunT:

$$\begin{aligned} E &\rightarrow TE' \\ E' &\rightarrow \ \mid +E \end{aligned}$$

- Pareilpourlenon-terminalT.

Butd'uneanalysed'unegrammaire

- Détecterdespropriétésdesnon-terminauxdansune grammaires,etdesgrammaires.
- Exemple:non-terminauxatteignablesoupas,productifsou pas,pouvantengendrerlemotvideoupas?Grammaires LL(1),LL(k)oupas?
- Défis:récurrenceentredeslesnon-terminauxd'une grammaire.
- Danscertainscasonmaîtriseunedescenterécursive *surun motdonné* pourlaconstructiond'unedérivation-casdes grammairesLL(k).
- Dansuneanalysedelagrammaire,onapasdemotd'entrée fixedonnémaisons'intéresseàunepropriétégénéraledela grammaire.

Illustrationduproblèmed'unedescenterécursive

- Unnon-terminalNest *productif* s'ilyaunerègleN $\to$ telle quetouslesnon-terminauxdans sontproductifs.(C'esten particuliervraiquand necontientquedesterminaux.)
- Imaginezlesrèglessuivantesdelagrammaire:

$$\begin{aligned} A &\to BC \mid \\ B &\to Cb \\ C &\to Ac \end{aligned}$$

- Ilfautéviterunedescenterécursivequimènedansuneboucle: Aproductif? $\to$ Bproductif? $\to$ Cproductif? $\to$ Aproductif? $\to :::$

Calcul d'unpointfixe

- Notredéfinitiondeproductivitéestcorrecte,maisilfaut organiserlecalculdifféremment:
  - Audébut,onposequetoutnon-terminalNpourlequelexiste unerègleN $\to w$,où $w \in V_T^*$,estproductif.
  - Puis,s'ilyaunerègleN $\to$,oùonadéjàreconnutousles non-terminauxdans commeétantproductifs,alorsonpose queNestaussiproductif.
  - Ons'arrêtesionnepeuplusajouterd'informationdecette façon.
- Ils'agitd'unpointfixe!

Pointsfixes

- Enmaths:un *pointfixe* d'unefonction $f : D \to D$ estune valeur $x \in D$ telleque $f(x) = x$.
- Applicationà l'algorithmique:
  - ledomaine D estl'ensembledetouteslesaffectationspossibles àdesvariablesd'intérêt. Dansl'exemple:touteslesaffectationspossiblesde P dutype $V_N \to \texttt{boolean}$.
  - lafonction f estunemiseàjourdesvariables. Dansl'exemple:propagationdel'informationdeproductivité d'unnon-terminalàunautre.
  - onnecherchepasunpointfixequelconque,maison commenceavecunevaleurinitiale,puisonapplique f jusqu'à unpointfixe.

Calcul d'un point fixe

Nous utiliserons dans nos algorithmes la construction "do ... until X1,...,Xn fix". Exemple productivité:

```
forall N ∈ V_N :
  if exists (N → ) ∈ R tel que ∈ V_T*
  then P(N)=true
  else P(N)=false
do
  forall (N → ) ∈ R :
    if P(M) forall non −terminals Min
    then P(N)=true
until P fix
```


La construction do ... until ... fix

Le code

```
//code initialisation de X
do
  //code mise a jour de X
until X fix
```

peut être traduit vers:

```
//code initialisation de X
do
  Xold=X
  //code mise a jour de X
while(X!=Xold)
```


Autre exemple symboles atteignables

- Un non-terminal N est *atteignable* (à partir de l'axiome S) s'il existe une dérivation $S \rightarrow^* N$.
- Calcul des non-terminaux atteignables: transparent suivant.
- Une grammaire est *réduite* si
  - tous ses non-terminaux sont productifs et
  - tous ses non-terminaux sont atteignables.
- Dans la suite nous supposons que toutes les grammaires sont réduites.

Calcul des non-terminaux atteignables

Algorithme

Donnée une grammaire $G = (V_T; V_N; S; R)$.

```
forall N ∈ V_N :
  A(N)=false
A(S)=true
do
  forall (N → ) ∈ R :
    if A(N)
    then forall non −terminals Min :
      A(M)=true
until A fix
```

Lemma

Pour tout $N \in V_N$: $A(N) == \texttt{true}$ ssi N est atteignable.

Non-terminauxquipeuventproduire ϵ

Algorithme

Donnéeunegrammaire $G = (V_T; V_N; S; R)$.

EPS= $\{N \in V_N \mid (N \rightarrow \ ) \in R\}$
do
EPS=EPS $\cup \{N \in V_N \mid (N \rightarrow N_1 ::: N_n) \in R;$
$N_1;:::; N_n \in \text{EPS}\}$
untilEPSfix

Lemme

Pourtout $N \in V_N$: $N \in$ EPSssi $N \rightarrow^*$.

CalculdeEPSsurl'exemple

E	$\rightarrow$	TE′	E′	$\rightarrow$	|+E
T	$\rightarrow$	FT′	T′	$\rightarrow$	|*T
F	$\rightarrow$	(E) | i	S	$\rightarrow$	E EOF

- Initialisation: $EPS = \{\text{E}'; \text{T}'\}$
- C'estdéjàunpointfixe!

Commentcalculer $FIRST_1$ danslecasgénéral?

- ImaginezunerègleA $\rightarrow$ BC d E
- Touslesnon-terminauxpeuventaprioriengendrer .
- SiB $\notin EPS$:danscetterègle,seulement $FIRST_1$(B) peut contribueràà $FIRST_1$(A).
- SiB $\in EPS$: $FIRST_1$(C) peutaussicontribueràà $FIRST_1$(A).
- SiB $\in EPS$ etC $\in EPS$: d doitêtredans $FIRST_1$(A).
- Dansaucundescas, $FIRST_1$(E) nepeutcontribuercarilse trouvederrièreleterminal d.

Lecalculde $FIRST_1$ danslecasgénéral

- Pourdesraisonstechniques,ilestplusfaciledecalculer d'abordunevariantede $FIRST_1$ sans :

$$Fi(N) = \{a \in V_T | N \rightarrow^* aw; w \in V_T^*\}$$

- Puisonenobtient $FIRST_1(N)$ enutilisant EPS quenous avonsdéjàcalculé.

Calculde $FIRST_1$ danslecasgénéral

Algorithme

Donnéeunegrammaire $G = (V_T; V_N; S; R)$.

pourtout $N \in V_N$:
  Fi(N)= $\{a \in V_T \mid (N \to N_1 ::: N_n a\ ) \in R;$
  $N_1; ::: ; N_n \in \text{EPS}\}$
do
  pourtout $(N \to N_1 ::: N_n M\ ) \in R$
  telque $M \in V_N; N_1; ::: ; N_n \in \text{EPS}$:
    Fi(N)=Fi(N) $\cup$ Fi(M)
until Ffix

Lemme

Pourtout $N \in V_N$: $Fi(N) = FIRST_1(\text{N}) - \{\ \}$

AnalysedeDonnéesStructurées-Cours6
└AnalysedeGrammaires

CalculdeFisurl'exemple

E	$\to$	TE′	E′	$\to$	|+E
T	$\to$	FT′	T′	$\to$	|*T
F	$\to$	(E) | i	S	$\to$	E EOF

$EPS = \{\text{E}'; \text{T}'\}$

Calcul

	Initial	*Iter*1	*Iter*2	*Iter*3
S	$\emptyset$	$\emptyset$	$\emptyset$	{i; (}
E	$\emptyset$	$\emptyset$	{i; (}	{i; (}
E′	{+}	{+}	{+}	{+}
T	$\emptyset$	{i; (}	{i; (}	{i; (}
T′	{*}	{*}	{*}	{*}
F	{i; (}	{i; (}	{i; (}	{i; (}

AnalysedeDonnéesStructurées-Cours6
└AnalysedeGrammaires

Calculde $FIRST_1$ danslecasgénéral

PourtoutN $\in V_N$:

$$FIRST_i(\text{N}) = \begin{cases} Fi(N) \cup \{\ \} & \text{siN} \in EPS \\ Fi(N) & \text{sinon} \end{cases}$$

AnalysedeDonnéesStructurées-Cours6
└AnalysedeGrammaires

Calculde $FIRST_1$ surl'exemple

E	$\to$	TE′	E′	$\to$	|+E
T	$\to$	FT′	T′	$\to$	|*T
F	$\to$	(E) | i	S	$\to$	E EOF

$EPS = \{\text{E}'; \text{T}'\}$

Calcul

	Fi	$FIRST_1$
S	{i; (}	{i; (}
E	{i; (}	{i; (}
E′	{+}	{+; }
T	{i; (}	{i; (}
T′	{*}	{*; }
F	{i; (}	{i; (}

Calculde $FIRST_1$ danslecasgénéral

Onétendmaintenant $FIRST_1$ àdesmotsdeterminauxet non-terminaux:

- $FIRST_1(\ ) = \{\ \}$
- pourtout $a \in V_T$: $FIRST_1(a\ ) = \{a\}$
- pourtout $N \in V_N$:

$$FIRST_1(N\ ) = \begin{cases} FIRST_1(N) & \text{si } N \notin EPS \\ (FIRST_1(N) \setminus \{\ \}) \cup FIRST_1(\ ) & \text{si } N \in EPS \end{cases}$$

Nousavonsbesoindeplusd'information!

- Lecalculde $FIRST_1$ n'estplussuffisantpoursavoirquelle productionappliquer!
- Exemple:E $'\rightarrow$ | +E
  Sinousvoyons + alorsilfaututiliserladeuxièmealternative pourréécrireE $'$.Maisquandfaut-ilappliquerlapremière?
- Ilnousmanqueuneinformation:quelsontlessymboles terminauxquipeuvent *suivre* àunmotproduitparun non-terminal?

Lafonction $FOLLOW_k$

Définition

Soit $G = (V_T; V_N; S; R)$ unegrammaire, $k \in \mathbb{N}$.Lafonction $FOLLOW_k : V_N \rightarrow 2^{V_T^*}$ estdéfiniepar

$$FOLLOW_k(N) = \{w \mid S \rightarrow^* \quad N \ ; \ ; \quad \in (V_T \cup V_N)^*; w \in FIRST_k(\ )\}$$

Remarques

- $FOLLOW_k(N)$ estl'ensembledetouslesmotsdeterminaux delongueur k quipeuvent,dansdesmotsde $\mathcal{L}(G)$,suivreà unmotdérivéde N.

Calculde $FOLLOW_1$

Algorithme

Donnéeunegrammaire $G = (V_T; V_N; S; R)$.

forall $N \in V_N$:
  Fo(N)= $\{a \in V_T \mid (M \rightarrow ::: NN_1 ::: N_k a :::) \in R; N_1; ::: ; N_n \in$ EPS$\}$
forall $(M \rightarrow ::: NN_1 ::: N_k N' :::) \in R$
with $N_1; ::: ; N_k \in$ EPS:
  Fo(N)= $Fo(N) \cup Fi(N')$
do
  forall $(M \rightarrow ::: NN_1 ::: N_k) \in R$
  with $N_1; ::: ; N_n \in$ EPS:
    Fo(N)=Fo(N) $\cup$ Fo(M)
untilFofix

Lemme

Si (S $\rightarrow$ N EOF) $\in R$,alors $\forall N \in V_N$: $Fo(N) = FOLLOW_1(N)$

Calculde $FOLLOW_1$ surl'exemple

$$\begin{array}{llll} E & \to & TE' & E' \to \ \mid +E \\ T & \to & FT' & T' \to \ \mid *T \\ F & \to & (E) \mid i & S \to E\ \texttt{EOF} \end{array}$$

$EPS = \{E'; T'\}$ $Fi(E') = \{+\}$ $Fi(T') = \{*\}$

Calcul

	Initial	*Iter*1	*Iter*2
S	$\emptyset$	$\emptyset$	$\emptyset$
E	{EOF;)}	{EOF;)}	{EOF;)}
E′	$\emptyset$	{EOF;)}	{EOF;)}
T	{+}	{+; EOF;)}	{+; EOF;)}
T′	$\emptyset$	{+}	{EOF;); +}
F	{*}	{*; EOF;); +}	{*; EOF;); +}

Commentchoisirlarègledansl'analyselexicale

Soit $A \to \ _1 \mid ::: \mid \ _n$ unealternative.Ilydeuxcas:

1. Soit,aucundes $FIRST_1(\ _i)$ necontient : commeavant:
   - Onchoisitlarègle $A \to \ _i$ quandlesymbolesuivantestdans $FIRST_1(\ _i)$ (ilssonttousdisjoints).
   - Erreursiaucuntel i existe
2. Soit $\ _i$ avec $\in FIRST_i(\ _i)$:
   - silesymbolesuivantestdans $FIRST_1(\ _j)$:choisir $A \to \ _j$, pour1 $\leq j \leq n$.
   - silesymbolesuivantestdans $FOLLOW_1(A)$:choisir $A \to \ _i$.
   - sinonErreur.

Finalement(!):lecritèrepourêtreLL(1)

Théorème

Lagrammaire $G = (V_T; V_N; S; R)$ estLL(1)ssipourtoutesles alternatives $A \to \ _1 \mid ::: \mid \ _n$:

- $FIRST_1(\ _1); ::: ; FIRST(\ _n)$ sontdisjointsentr'eux;
- Si $\in FIRST_1(\ _i)$,alorspourtous $j \neq i$:

$$FIRST_1(\ _j) \cap FOLLOW_1(A) = \emptyset$$

Remarque

Condition(1)impliquequ'auplusundesensembles $FIRST_1(\ _i)$ contient .

AnalysedeDonnéesStructurées-Cours7

RalfTreinen

UniversitéParisDiderot
UFRInformatique
LaboratoirePreuves,ProgrammesetSystèmes

treinen@pps.univ-paris-diderot.fr

18mars2015

Vuauderniercours

- Calculd'unpointfixepar `do...untilXfix`
- Définition: $Fi\ N \quad \{a \in V_T \mid N \rightarrow^* a\alpha\}$
- Onaque $Fi\ N \quad FIRST_1\ N \setminus \{\epsilon\}$
- Algorithmepourlecalculde $Fi\ N$ enprésencederègles $M \rightarrow \epsilon$
- Resteàfaire:calculde $FIRST_1\ \alpha$ pour $\alpha \in V_T \cup V_N{}^*$

Rappel:Calculde *Fi*

Algorithme

Donnéeunegrammaire $G \quad V_T, V_N, S, R$.

pourtout $N \in V_N$:
  Fi(N)= $\{a \in V_T \mid N \rightarrow N_1 \ldots N_n a\alpha \in R, N_1, \ldots, N_n \in \text{EPS}\}$

do
  pourtout $N \rightarrow N_1 \ldots N_n M\alpha \in R$
  telque $M \in V_N, N_1, \ldots, N_n \in \text{EPS}$:
    Fi(N)=Fi(N) $\cup$ Fi(M)
until Fifix

Exemple

Grammaire

$$\begin{array}{llll} A \rightarrow \epsilon \mid a & & B \rightarrow \epsilon \mid Ab \\ C \rightarrow ABc & & D \rightarrow BCd \end{array}$$

Non-terminauxquipeuventproduire ϵ

EPS {A, B}

Calcul

	Initial	*Iter*1	*Iter*2
A	{ }	{ }	{ }
B	{b}	{ , b}	{ , b}
C	{c}	{ , b, c}	{ , b, c}
D	∅	{b, c}	{ , b, c}

Calculde $FIRST_1$ danslecasgénéral

PourtoutN $\in V_N$:

$$FIRST_i\ \mathrm{N} \quad \begin{cases} Fi\ N \cup \{\epsilon\} & \text{siN} \in EPS \\ Fi\ N & \text{sinon} \end{cases}$$

Calculde $FIRST_1$ surl'exemple

$$\begin{array}{llllll} \mathrm{E} & \to & \mathrm{TE'} & \mathrm{E'} & \to & \epsilon \mid \mathtt{+E} \\ \mathrm{T} & \to & \mathrm{FT'} & \mathrm{T'} & \to & \epsilon \mid \mathtt{*T} \\ \mathrm{F} & \to & \mathtt{(E)} \mid \mathtt{i} & \mathrm{S} & \to & \mathrm{E}\ \mathtt{EOF} \end{array}$$

$$EPS \quad \{\mathrm{E'}, \mathrm{T'}\}$$

Calcul

	Fi	$FIRST_1$
S	{i, (}	{i, (}
E	{i, (}	{i, (}
E′	{+}	{+, ϵ}
T	{i, (}	{i, (}
T′	{*}	{*, ϵ}
F	{i, (}	{i, (}

Calculde $FIRST_1$ danslecasgénéral

Onétendmaintenant $FIRST_1$ àdesmotsdeterminauxet non-terminaux:

- $FIRST_1\ \epsilon \quad \{\epsilon\}$
- pourtout $a \in V_T$: $FIRST_1\ a\alpha \quad \{a\}$
- pourtout $N \in V_N$:

$$FIRST_1\ N\alpha \quad \begin{cases} FIRST_1\ N & \text{si } N \notin EPS \\ FIRST_1\ N \setminus \{\epsilon\} \cup FIRST_1\ \alpha & \text{si } N \in EPS \end{cases}$$

Calculde $FIRST_1$ descôtésdroitesdansl'exemple

$FIRST_1$ TE′	{i, (}
$FIRST_1\ \epsilon$	{ϵ}
$FIRST_1$ + E	{+}
$FIRST_1$ FT′	{i, (}
$FIRST_1$ * T	{*}
$FIRST_1$ (E)	{(}
$FIRST_1$ i	{i}
$FIRST_1$ S EOF	{i, (

Nousavonsbesoindeplusd'information!

- Lecalculde $FIRST_1$ n'estplussuffisantpoursavoirquelle productionappliquer!
- Exemple:E $'\to \epsilon \mid$ +E
  Sinousvoyons + alorsilfaututiliserladeuxièmealternative pourréécrireE $'$.Maisquandfautilappliquerlapremière?
- Ilnousmanqueuneinformation:quelsontlessymboles terminauxquipeuvent *suivre* àunmotproduitparun non-terminal?

Lafonction $FOLLOW_k$

Définition

Soit G V_T, V_N, S, R unegrammaire, $k \in \mathbb{N}$.Lafonction $FOLLOW_k$ $V_N \to 2^{V_T^*}$ estdéfiniepar

$FOLLOW_k$ N $\{w \mid S \to^* \beta N \gamma, \beta, \gamma \in V_T \cup V_N\ ^*, w \in FIRST_k\ \gamma\ \}$

Remarques

- $FOLLOW_k$ N estl'ensembledetouslesmotsdeterminaux delongueur k quipeuvent,dansdesmotsde $\mathcal{L}$ G ,suivreà unmotdérivéde N.

Calculde $FOLLOW_1$

Algorithme

Donnéeunegrammaire G V_T, V_N, S, R .

for all $N \in V_N$:
  Fo(N)= $\{a \in V_T \mid M \to \ldots NN_1 \ldots N_k a \ldots \in R, N_1, \ldots, N_n \in$ EPS$\}$
for all $M \to \ldots NN_1 \ldots N_k N^\emptyset \ldots \in R$
with $N_1, \ldots, N_k \in$ EPS:
  Fo(N)= $Fo\ N \cup Fi\ N^\emptyset$
do
  for all $M \to \ldots NN_1 \ldots N_k \in R$
  with $N_1, \ldots, N_n \in$ EPS:
    Fo(N)=Fo(N) $\cup$ Fo(M)
until Fofix

Lemme

Si S $\to$ N EOF $\in R$,alors $\forall N \in V_N$: $Fo\ N$ $FOLLOW_1$ N

Calculde $FOLLOW_1$ surl'exemple

E $\to$ TE $'$ E$'$ $\to$ $\epsilon \mid$ +E
T $\to$ FT $'$ T$'$ $\to$ $\epsilon \mid$ *T
F $\to$ (E) $\mid$ i S $\to$ E EOF

EPS $\{$E$', T'\}$ Fi E$'$ $\{+\}$ Fi T$'$ $\{*\}$

Calcul

	Initial	*Iter*1	*Iter*2
S	$\emptyset$	$\emptyset$	$\emptyset$
E	{EOF,)}	{EOF,)}	{EOF,)}
E$'$	$\emptyset$	{EOF,)}	{EOF,)}
T	{+}	{+, EOF,)}	{+, EOF,)}
T$'$	$\emptyset$	{+}	{EOF,), +}
F	{*}	{*, +}	{*, EOF,), +}

Finalement(!):lecritèrepourêtreLL(1)

Théorème

Lagrammaire G V_T, V_N, S, R estLL(1)ssipourtoutesles alternatives $A \to \alpha_1 \mid \ldots \mid \alpha_n$:

- $FIRST_1\ \alpha_1, \ldots, FIRST_1\ \alpha_n$ sontdisjointsentr'eux;
- Si $\epsilon \in FIRST_1\ \alpha_i$,alorspourtous $j \neq i$:

$$FIRST_1\ \alpha_j \cap FOLLOW_1\ A \quad \emptyset$$

Remarque

Condition(1)impliquequ'auplusundesensembles $FIRST_1\ \alpha_i$ contient ϵ.

Lecritèresurl'exemple

Non-t rminal	Cas1	$FIRST_1$	Cas2	$FIRST_1$
E	TE′	{i, (}		
E′	ϵ	{ϵ}	+ E	{+}
T	FT′	{i, (}		
T′	ϵ	{ϵ}	* T	{*}
F	(E)	{(}	i	{i}
S	E EOF	{i, (}		

- $FOLLOW_1$ E′ {), EOF} disjointavec {+} ☺
- $FOLLOW_1$ T′ {EOF,), +} disjointavec {*} ☺
- {(} disjointavec {i} ☺
- Conclusion:lagrammaireestLL(1)!

Commentchoisirlarègledansl'analysesyntaxique

Soit $A \to \alpha_1 \mid \ldots \mid \alpha_n$ unealternative.Ilydeuxcas:

1. Soit,aucundes $FIRST_1\ \alpha_i$ necontient ϵ : commeavant:
   - Onchoisitlarègle $A \to \alpha_i$ quandlesymbolesuivantestdans $FIRST_1\ \alpha_i$ (ilssonttousdisjoints).
   - Erreursiaucuntel i existe
2. Soit α_i avec $\epsilon \in FIRST_i\ \alpha_i$:
   - silesymbolesuivantestdans $FIRST_1\ \alpha_j$:choisir $A \to \alpha_j$, pour1 $\leq j \leq n$.
   - silesymbolesuivantestdans $FOLLOW_1\ A$:choisir $A \to \alpha_i$.
   - sinonErreur.

Lecodedul'analyseurdesyntaxesurl'exemplel

```
import java.io.*;

/* LL(1)parserforexpressionswithpriorities,                    */
/* terminatedbythesymbol'$'                                    */
class Parser{

    protected LookAhead1Readerreader;

    public Parser(LookAhead1Readerr){
        reader=r;
    }

    publicvoid term( char c) throws IOException,ReadException{
        /* consumethecharacterc                                */
```

Lecodedul'analyseurdesyntaxesurl'exemple II

```
        reader.eat(c);
    }

    public void nonterm_S() throws IOException,ReadException{
        /* S ->E$ */
        this.nonterm_E();
        this.term('$');
    }

    public void nonterm_E() throws IOException,ReadException{
        /* E ->TE1 */
        this.nonterm_T();
        this.nonterm_E1();
    }
```


Lecodedul'analyseurdesyntaxesurl'exemple III

```
    public void nonterm_E1() throws IOException,ReadException{
        if (reader.check('$')||reader.check(')')){
            /* E1 ->epsilon */
        } elseif (reader.check('+')){
            /* E1 ->+E */
            this.term('+');
            this.nonterm_E();
        }
    }

    public void nonterm_T() throws IOException,ReadException{
        /* T ->FT1 */
        this.nonterm_F();
        this.nonterm_T1();
    }
```


Lecodedul'analyseurdesyntaxesurl'exemple IV

```
    public void nonterm_T1() throws IOException,ReadException{
        if (reader.check('$')||reader.check(')')||reader.check('+')){
            /* T1 ->epsilon */
        } elseif (reader.check('*')){
            /* T1 -> * T */
            this.term('*');
            this.nonterm_T();
        }
    }

    public void nonterm_F() throws IOException,ReadException{
        if (reader.check('(')){
            /* F ->(E) */
            this.term('(');
```


Lecodedul'analyseurdesyntaxesurl'exemple V

```
            this.nonterm_E();
            this.term(')');
        } elseif (reader.check('i')){
            /* F ->i */
            this.term('i');
        }
    }
}
```

Syntaxeconcrète

- *Syntaxeconcrète* :c'estladéfinitiondelaformecorrectede l'écriture(représentationtextuelle).Elleestnormalement définiepardesexpressionsrégulières,etunegrammaire hors-contexte.
- Lavérificationqu'untexted'entréesuitlasyntaxeconcrète d'unlangageestréaliséeencoopérationparl'analyselexicale etl'analysesyntaxique.
- Or,normalementonneveutpasseulementsavoirsavoirsi l'entréeestcorrecteoupas,maisaussifairequelquechose avec,parexemple:
  - DocumentXMLreprésentantdesdonnéesgéographiques: afficherunecarte.
  - Programmeécritdansunlangagedeprogrammation: l'exécuter(casd'uninterpréteur),ouengendrerducode exécutable(casd'uncompilateur).

- Onvadonc construireunereprésentationdelastructureque lesanalyseslexicalesetsyntaxiquesontdécouvertes:c'estla *syntaxeabstraite* .
- Lepassagedelasyntaxeconcrèteàlasyntaxeabstraiteadeux fonctions:
  - Vérifierquel'entréecorrespondauxrèglesdelasyntaxe concrète,
  - sic'estlecas,construireunereprésentationenformedela syntaxeabstraite.
- Ilyaicisouventune *abstraction* :certainsdétailsdela représentationtextuelle(ensyntaxeconcrète)sontomisdans lasyntaxeabstraitecarilsnesontpaspertinentspourlasuite.
- Onessayed'abstrairedetoutquin'estpasnécessairepourla suite:simplifiertantquepossible!

Abstraction

- Quelssontlesdétailsdelasyntaxeconcrètedontonpeutfaire abstraction:çadépenddecequ'oncomptefaireavec!
- Nousavonsdéjàvucertainesabstractionsfaitesparl'analyse lexicale,dansle contextedeslangagesdeprogrammation:
  - lesespaces(maisattentionauxcasoùl'incrémentationdes lignesindiquelastructureduprogramme)
  - lescommentaires(maisattentionauxcasoùlescommentaires sontimportantespourlasuite,parexemplesionécritun *prettyprinter* decodesource)
- Dansunpremiertempsonpeututiliserl'arbrededérivation construitparl'analysesyntaxiquecommesyntaxeabstraite.

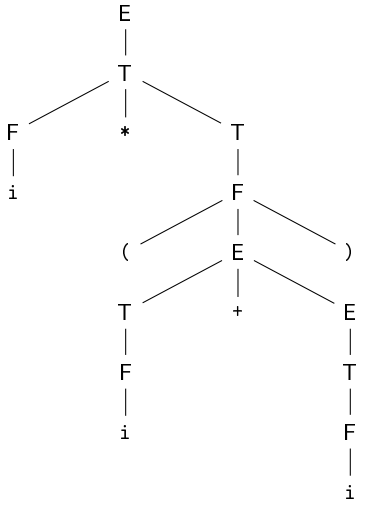
Arbrededérivationvs.Syntaxeabstraite

- Engénéral,l'arbrededérivationcontienttropd'information.
- Raison:l'arbrededérivationdit *comment* lastructureaété découvertedansletexted'entrée.Or,c'estseulementla structureelle-mêmequiestpertinentepourlasuite.
- Exemple:grammaire

$$\begin{aligned} E &\rightarrow E{+}T \mid T \\ T &\rightarrow T{*}F \mid F \\ F &\rightarrow (E) \mid \texttt{i} \end{aligned}$$

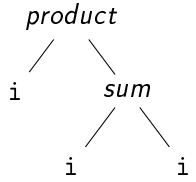
- Arbrededérivationpour $i * i + i$:voirleurlispareilsurtableau

Arbrededérivationpour $i * (i + i)$

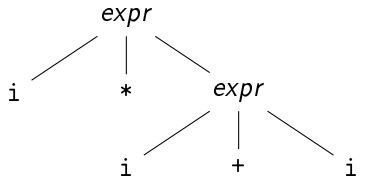


Quelinformationretenirdanslasyntaxeabstraite?

- LadistinctionentreE,TetFn'estpluspertinente.
- Lesparenthèsesnesontpluspertinentesnonplus:elles serventàindiquerlastructuredanslasyntaxeconcrète,mais unefoislastructuredécouverteelleneserventplusàrien.
- Structureàretenir:



Autrepossibilitépourlasyntaxeabstraite



- Icionaurauneseuleclasse,avecl'opérateurcommeargument supplémentaireduconstructeur.
- Solutionàpréférersidanslasuitetouslesopérateurssont traitésdelamêmefaçon.

DéfinitiondelaSyntaxeAbstraitesurl'exemple1

```
abstract class Expression{
    public boolean isIdentifier(){
        return false;
    }
    public boolean IsSum(){
        return false;
    }
    public boolean IsProduct(){
        return false;
    }
    abstract void print();

}
```

DéfinitiondelaSyntaxeAbstraitesurl'exempleII

```
class Identifier extends Expression{
    public boolean isIdentifier(){
        return true ;
    }
    public Identifier(){
    }
    public void print(){
        System.out.print("i");
    }
}

class Sum extends Expression{
    private Expression left;
    private Expression right;
    public boolean isSum(){
```


DéfinitiondelaSyntaxeAbstraitesurl'exempleIII

```
        return true ;
    }
    public Sum(Expression e1,Expression e2){
        left=e1;
        right=e2;
    }
    public void print(){
        System.out.print("Sum(");
        left.print();
        System.out.print(',');
        right.print();
        System.out.print(")");
    }
}
```


DéfinitiondelaSyntaxeAbstraitesurl'exempleIV

```
class Product extends Expression{
    private Expression left;
    private Expression right;
    public boolean isProduct(){
        return true ;
    }
    public Product(Expression e1,Expression e2){
        left=e1;
        right=e2;
    }

    public void print(){
        System.out.print("Product(");
        left.print();
        System.out.print(',');
```


DéfinitiondelaSyntaxeAbstraitesurl'exempleV

```
        right.print();
        System.out.print(")");
    }

}
```

ConstructiondelaSyntaxeAbstraitesurl'exemplel

```
import java.io.*;

/* LL(1)parserforexpressionswithpriorities, */
/* terminatedbythesymbol'$' */
class Parser{

    protected LookAhead1Readerreader;

    public Parser(LookAhead1Readerr){
        reader=r;
    }

    publicvoid term(char c) throws IOException,ReadException{
        /* consumethecharacterc */
```


ConstructiondelaSyntaxeAbstraitesurl'exemplell

```
        reader.eat(c);
    }

    public Expressionnonterm_S()
        throws IOException,ReadException,ParserException{
        /* S ->E$ */
        Expressione= this.nonterm_E();
        this.term('$');
        return e;
    }

    public Expressionnonterm_E()
        throws IOException,ReadException,ParserException{
        /* E ->TE1 */
        Expressione1= this.nonterm_T();
```


ConstructiondelaSyntaxeAbstraitesurl'exemplelll

```
        Expressione2= this.nonterm_E1();
        if (e2==null){
            return e1;
        } else {
            returnnew Sum(e1,e2);
        }
    }

    public Expressionnonterm_E1()
        throws IOException,ReadException,ParserException{
        if (reader.check('$')||reader.check(')')){
            /* E1 ->epsilon */
            returnnull;
        } elseif (reader.check('+')){
            /* E1 ->+E */
```


ConstructiondelaSyntaxeAbstraitesurl'exemplelV

```
            this.term('+');
            returnthis.nonterm_E();
        } else {
            thrownew ParserException("cannot␣reduce␣E'");
        }
    }

    public Expressionnonterm_T()
        throws IOException,ReadException,ParserException{
        /* T ->FT1 */
        Expressione1= this.nonterm_F();
        Expressione2= this.nonterm_T1();
        if (e2==null){
            return e1;
        } else {
```

ConstructiondelaSyntaxeAbstraitesurl'exempleV

```
                returnnew      Product(e1,e2);
            }
        }

        public    Expressionnonterm_T1()
            throws   IOException,ReadException,ParserException{
            if   (reader.check('$')||reader.check(')')||reader.check('+')){
                /*  T1 ->epsilon      */
                returnnull      ;
            }  elseif      (reader.check('      *')){
                /*  T1 -> *  T  */
                this  .term('   *');
                returnthis      .nonterm_T();
            }  else   {
                thrownew      ParserException("cannot␣reduce␣T");
```


ConstructiondelaSyntaxeAbstraitesurl'exempleVI

```
            }
        }

        public    Expressionnonterm_F()
            throws   IOException,ReadException,ParserException{
            if   (reader.check('(')){
                /*  F ->(E)      */
                this  .term('(');
                Expressione=          this  .nonterm_E();
                this  .term(')');
                return    e;
            }  elseif      (reader.check('i')){
                /*  F ->i    */
                this  .term('i');
                returnnew      Identifier();
```


ConstructiondelaSyntaxeAbstraitesurl'exempleVII

```
            }  else   {
                thrownew      ParserException("cannot␣reduce␣F");
            }
        }
    }
```

AnalysedeDonnéesStructurées-Cours8

RalfTreinen

UniversitéParisDiderot
UFRInformatique
LaboratoirePreuves,ProgrammesetSystèmes

treinen@pps.univ-paris-diderot.fr

25mars2015

Formatsgénériquespourdesdonnéesstructurées

- Permettentlareprésentationtextuelledededonnéesstructurées
- Utilespour:
  - Stockerdesdonnéesdansunfichier:persistance
  - Transmissiondedonnéesentredesapplicationshétérogènes (éventuellementécritesdansdeslangagesdifférents)
  - Transmissiondedonnéessurleweb(parexemple,AJAX)
  - Desfichiersdeconfiguration(carfacileàmodifierpourdes humains,etfacileàinterpréterpourdesprogrammes)
- ExemplesdanscecoursJSONetXML

LeformatJSON

- **J**ava**S**cript **O**bject **N**otation
- Ledocumentdestandardisationcommenceaveclerésumé suivant:

  JavaScriptObjectNotation(JSON)isalightweight, text-based,language-independentdatainterchange format.
  [...]
  JSONdefinesasmallsetofformattingrulesforthe portablerepresentationofstructureddata.

LeformatJSON

- Valeursdebase: `true`, `f lse`, `null`,valeursnumériques,et chaînesdecaractères.
- Deuxmécanismespourcombinerdesvaleurs:
  - *array* :c'estsimplementunelistedevaleursentrecrochet[et],séparéespardesvirgules
  - *object* :c'estuneséquencedepaires,chacuneconsistanten unechaînedecaractères,etunevaleur.Laséquenceestnotée entreaccolades{et},lespairessontséparéespardesvirgules.
- Attentionàladifférenceentre *terminé* et *séparé* pardes virgules.

Exemple

```
{
   "menu":{
      "id":"file",
      "value":"File",
      "popup":{
         "menuitem":[
            {"value":"New","onclick":"CreateNewDoc()"},
            {"value":"Open","onclick":"OpenDoc()"},
            {"value":"Close","onclick":"CloseDoc()"}
         ]
      }
   }
}
```

Source: `http://fr.wikipedia.org/wiki/JavaScript_Object_Notation`

Unautreexemple

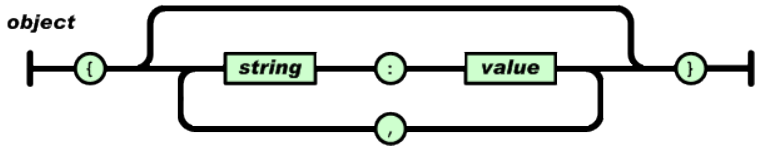
```
{
  "Image":{
     "Width":800,
     "Height":600,
     "Title":"View␣from␣15th␣Floor",
     "Thumbnail":{
        "Url": "http://www.example.com/image/481989",
        "Height":125,
        "Width":100
     },
     "Animated":false,
     "IDs":[116,943,234,38793]
  }
}
```

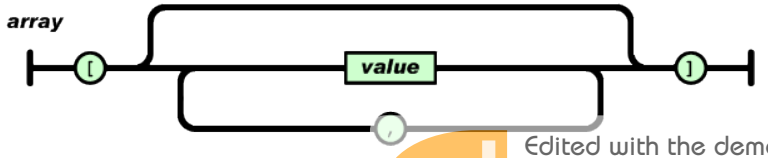
Source: `http://tools.ietf.org/html/rfc7159`

Ladéfinitionofficielle

- Ontrouvesurle“siteofficiel”deJSON, `http://json.org/`, unesériedediagrammesdesyntaxe.
- Ilyaaussiunepropositiondestandardisationdelapartdela IETF(InternetEngineeringTaskForce)“RFC7159”,disponible àl’adresse `http://tools.ietf.org/html/rfc7159`.

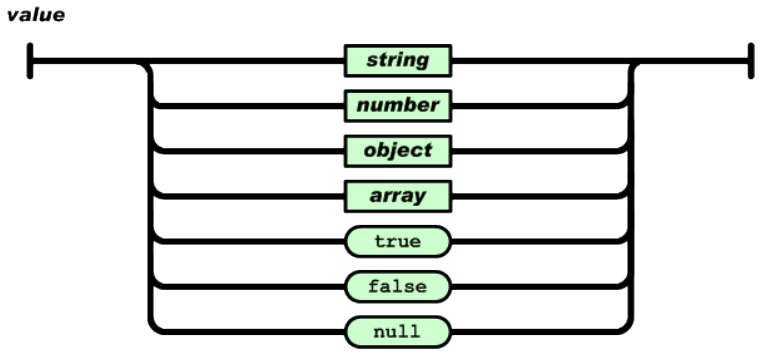
Diagrammesdesyntaxe





Source: `http://json.org/`

Diagrammesdesyntaxe

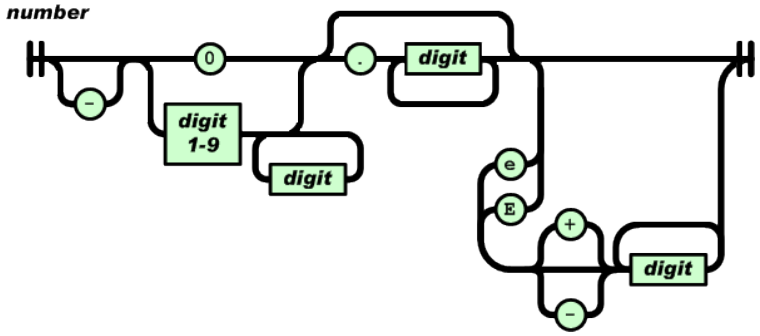


Source: http://json.org/

Diagrammesdesyntaxe

Source: http://json.org/

Diagrammesdesyntaxe



Source: http://json.org/

Commentfaireanalyselexicale/syntaxique?

- Expressionsrégulièrespourleschaînesetlesvaleurs numériques.
- Jetons(délivrésparl'analyselexicale):
  - `string`, `number`,(munisdevaleurs)
  - `true`, `false`, `null`,
  - `,` , `:`, `{`, `}`, `[`, `]`

Lagrammaire(premièreretentative)

- $V_N = \{\text{Value}, \text{Object}, \text{Array}, \text{Aseq}, \text{Oseq}, \text{AseqNV}, \text{OseqNV}\}$
- Axiome:value
- Règles:

$$
\begin{array}{rcl}
\text{Value} & \rightarrow & \texttt{string} \mid \texttt{number} \mid \text{Object} \mid \text{Array} \mid \texttt{true} \mid \texttt{f lse} \mid \texttt{null} \\
\text{Array} & \rightarrow & \texttt{[}\ \text{Aseq}\ \texttt{]} \\
\text{Aseq} & \rightarrow & \epsilon \mid \text{AseqNV} \\
\text{AseqNV} & \rightarrow & \text{Value} \mid \text{Value}\ \texttt{,}\ \text{AseqNV} \\
\text{Object} & \rightarrow & \texttt{\{}\ \text{Oseq}\ \texttt{\}} \\
\text{Oseq} & \rightarrow & \epsilon \mid \text{OseqNV} \\
\text{OseqNV} & \rightarrow & \texttt{string:}\ \text{Value} \mid \texttt{string:}\ \text{Value}\ \texttt{,}\ \text{Oseq}
\end{array}
$$

- Est-ceLL(1)?

Lagrammaire(deuxièmetentative)

- $V_N = \{\text{Value}, \text{Object}, \text{Array}, \text{Aseq}, \text{Oseq}, \text{Aseq}', \text{Oseq}'\}$
- Axiome:value
- Règles:

$$
\begin{array}{rcl}
\text{Value} & \rightarrow & \texttt{string} \mid \texttt{number} \mid \text{Object} \mid \text{Array} \mid \texttt{true} \mid \texttt{f lse} \mid \texttt{null} \\
\text{Array} & \rightarrow & \texttt{[}\ \text{Aseq}\ \texttt{]} \\
\text{Aseq} & \rightarrow & \epsilon \mid \text{ValueAseq}' \\
\text{Aseq}' & \rightarrow & \epsilon \mid \texttt{,}\ \text{ValueAseq}' \\
\text{Object} & \rightarrow & \texttt{\{}\ \text{Oseq}\ \texttt{\}} \\
\text{Oseq} & \rightarrow & \epsilon \mid \texttt{string:}\ \text{ValueOseq}' \\
\text{Oseq}' & \rightarrow & \epsilon \mid \texttt{,string:}\ \text{ValueOseq}'
\end{array}
$$

- Est-cemaintenantLL(1)?

AnalyserundocumentécritenJSON

- Ilestdoncassezfaciled'écrireunanalyseursyntaxiquepourle formatJSON.
- Ilexisteaussidenombreusesbibliothèques,pourpresquetoutes leslangagesdeprogrammation(voirlalistesur `json.org`).

LeformatXML

- **E**xtensibe **M**arkup **L**anguage (fr.: *langagedebalisageextensible*)
- Objectifs:lesmêmesquepourJSON
- Ils'agitd'uneinstanced'unformatplusgénéral,dunom SGML(**S**tandard **G**eneralized **M**arkup **L**anguage).
- LangageplusrichequeJSON,maisaussiplusverbeuxet moinsélégant.
- LelangageXMLeststandardiséparleW3C(WorldWideWeb Consortium).

La structure d'un document XML

- Un document XML décrit un arbre.
- Un nœud peut avoir un nombre arbitraire d'enfants.
- Un nœud peut avoir des *attributs*.
- Les enfants d'un nœud sont appelés ses *éléments* ; un élément peut être un morceau de texte, ou un autre nœud.

Exemple

Le même exemple que dans la section sur JSON :

```
menu id="file" value="File">
  popup>
    menuitem value="New" onclick="CreateNewDoc()">
    /menuitem>
    menuitem value="Open" onclick="OpenDoc()">
    /menuitem>
    menuitem value="Close" onclick="CloseDoc()">
    /menuitem>
  /popup>
/menu>
```

(sans s'utiliser le raccourci pour les nœuds sans enfants)

Remarques

- Ça ressemble à du HTML, et pour cause : XML et HTML sont tous les deux des dérivés du langage plus général SGML.
- L'utilisation des chevrons < et > et assez caractéristique pour ces langages.
- Le XML est plus strict que le HTML, par exemple :
  - XML exige que les valeurs des attributs soient écrites entre guillemets, contrairement aux premières versions de HTML.
  - XML exige que toutes les balises soient proprement fermées, contrairement à HTML qui est très libéral en ce regard.

Vers une définition de la syntaxe

- Un nœud commence avec une balise (angl. : *tag*) de la forme

  <*m*>

  et se termine avec

  </*m*>

  nom est mot quelconque, appelé le *nom* de cet élément.
- Le nom est obligatoirement le même dans la balise de début et de fin du même nœud. Il est permis d'utiliser le même nom pour plusieurs nœuds du même arbre.
- Une balise de début peut en plus contenir des spécifications de attributs avec leurs valeur.

Versunedéfinitiondelasyntaxe

- Unepaireattribut/valeurestdonnéedanslaforme

  attr=valeur

  où *attr* estunmot(appelél'attribut),et *valeur* estunechaîne decaractèresentreguillemets(appeléelavaleur).
- Iln'estpaspermisdéfinirdanslamêmebalisedeuxfoisle mêmeattribut.

Raccourcipourdesnœudssansenfants

Lasyntaxe

<nomattr1="value1"...attrn="valuen"/>

estunraccourcipour

<nomattr1="value1"...attrn="valuen">
</nom>

Exemple

Lemêmeexempleavecceraccourci:

```
menuid="file"value="File">
  popup>
    menuitemvalue="New"onclick="CreateNewDoc()"/>
    menuitemvalue="Open"onclick="OpenDoc()"/>
    menuitemvalue="Close"onclick="CloseDoc()"/>
  /popup>
/menu>
```

Source: `http://fr.wikipedia.org/wiki/JavaScript_Object_Notation`

Exemple:DonnéesOpenStreetMap

```
<?xmlversion="1.0"encoding="UTF-8"?>
<osmversion="0.6"
     copyright="OpenStreetMapandcontributors"
     license="http://opendatacommons.org/licenses/odbl/1-0/">
  <wayid="62378611"
       visible="true"version="8"changeset="20691652"
       timestamp="2014-02-21T11:23:38Z"user="thibdrev"uid="1279506">
    <ndref="779143878"/>
    <ndref="2198721646"/>
    <ndref="2198721727"/>
    ........
    <tagk="amenity"v="university"/>
    <tagk="building"v="yes"/>
    <tagk="name"v="HalleauxFarines(UniversitéParisDiderot)"/>
    <tagk="source"v="cadastre-dgi-frsource:DirectionGénéraledesImpôts-Cadastre.
    <tagk="wheelchair"v="yes"/>
    <tagk="wikipedia"v="fr:UniversitéParisVII-Diderot"/>
  </way>
</osm>
```

UnegrammairepourXML?

- Terminaux(jetons): `mot`, `string`, `texte`, `/`, `=`, `<`, `>`
- Non-terminaux:Arbre,Start,End,Attrbs,Elements
- Axiome:Arbre
- Règles:

$$
\begin{aligned}
\text{Arbre} &\rightarrow \text{StartElementsEnd} \\
\text{Start} &\rightarrow \texttt{<mot}\ \ \text{Attrbs}\ \texttt{>} \\
\text{End} &\rightarrow \texttt{</mot>} \\
\text{Attrbs} &\rightarrow \epsilon \mid \texttt{mot=string}\ \ \text{Attrbs} \\
\text{Elements} &\rightarrow \epsilon \mid \text{ArbreElements} \mid \texttt{texte}\ \text{Elements}
\end{aligned}
$$

AnalysedeDonnéesStructurées-Cours8
└LeformatXML

UnegrammairepourXML?

- Attention:Cettegrammairenedéfinitpascomplètementle langageXML.
- Ilyadeuxélémentsdeladéfinitionquinesontpasexprimés parlagrammaire:
  - Toutebalisededébutdoitêtrefermée*par*unebalisedefin *du mêmenom* .
  - Touslesattributsdansunebalisededébutdoiventêtre *différents*.
- Peutonaméliorerlagrammairepourexprimeraussicesdeux restrictions?
- Réponse:non!voirlestransparentssuivants.

AnalysedeDonnéesStructurées-Cours8
└LeformatXML

Versunepreuve

- Nousallonsmontrerquec'estimpossibledédéfinirlelangage XMLentièrementparunegrammairehors-contexte.
- Celaimpliqueraquec'estégalementimpossibleonutilisant unecombinaisonexpressionsrégulières-grammaire hors-contexte,cartouteexpressionrégulièrepeutêtre transforméeenunegrammaire.

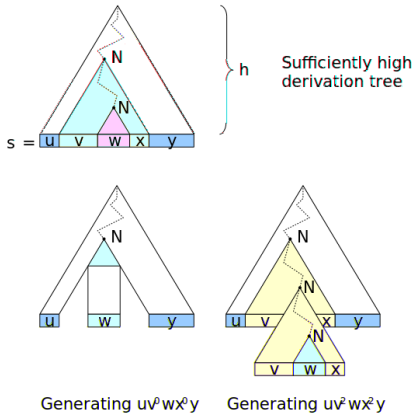
AnalysedeDonnéesStructurées-Cours8
└LeformatXML

Rappel:lemmedepompagepourleslangagesalgébriques

Soit L unlangagealgébrique.Alorsilexisteun $p \geq 0$telquetout mot $s \in L$ avec $|s| \geq p$ peutêtredécomposéen $z = uvwxy$ avec:

- $|vwx| \leq p$
- $|vx| \geq 1$
- $uv^iwx^iy \in L$ pourtout $i \geq 0$

Idéedelapreuvedulemmedepompage



Source:JochenBurghardt

ApplicationdulemmedepompageàXML(1)

Considéronslemotsuivantquiesten L_{XML} :

$$s = <\underbrace{a\cdots a}_{p}\underbrace{b\cdots b}_{p}></\underbrace{a\cdots a}_{p}\underbrace{b\cdots b}_{p}>$$

- Onpeutécrire $s = uvwxy$,avec $|vwx| \leq p$ et $|vx| \geq 1$, telque $uwy \in L_{XML}$.
- Cas1: vwx estentièrementdanslapartieverteoulapartie rouge:contradictioncar,aprèspassageà uwy,lesparties verteetrougesontdelongueurdifférente!

ApplicationdulemmedepompageàXML(2)

$$s = <\underbrace{a\cdots a}_{p}\underbrace{b\cdots b}_{p}></\underbrace{a\cdots a}_{p}\underbrace{b\cdots b}_{p}>$$

- $s = uvwxy$, $|vwx| \leq p$, $|vx| \geq 1$, $uwy \in L$.
- Cas2: vwx

Pourallerplusloin

- Ilexisteunlangagede *schéma* pourdesdocumentsXML.Un schémarestreintlaformed'undocumentXML.DTD (**D**ocument **T**ype **D**efinition).
- Unschémapeutparexempledéfinirquelsattributssont obligatoires(autorisés)pourqueltype,ouquelstypes d'élémentssontobligatoires(autorisés)pourunnœudd'un certaintype.
- Ilexisteaussideslangagesrichespourdesrequêtes(extraction dedonnéesd'undocumentXML),parex.XPath,XQuery.
- Pourensavoirplus:voirlecoursXMLduM1

AnalysedeDonnéesStructurées-Cours9

RalfTreinen

UniversitéParisDiderot
UFRInformatique
LaboratoirePreuves,ProgrammesetSystèmes

treinen@pps.univ-paris-diderot.fr

2avril2015

AnalyseSémantique

- Noussavonsmaintenantconstruireunarbredesyntaxe abstraiteàpartird'unereprésentationtextuelle(syntaxe concrète).
- Parexemple:syntaxeabstraitepourunlangagede programmation,pourunlangagededonnéesstructurées commeXMLouJSON.
- Nousavonsaussivulasemainedernièrequecertaines restrictionssytaxiquesnepeuventpasêtreréaliséesparune unegrammaire.
- C'estmaintenantlerôledel'analysesémantique.
- Lestechniquessontlesmêmesquepourl'interprétation.

ThéorieetPratique

Tâche	Modelthéorique	Réalisation
Analyselexicale	Expressionsrégulières	FichierJFlex
Analysesyntaxique	Grammaires	Implémentation d'un parseurLL(1)
Syntaxeabstraite	Définitioninductive	ClassesJava
Sémantique	Règlessémantiques	Parcoursd'arbre

DéfinitionInductive

L'ensemble E desarbresdesyntaxeabstraitereprésentantdes expressionsrégulièresestdéfinicommesuit:

- tout $Int\ n$,où $n \in \mathbb{N}$,estunélémentde E
- si $e_1, e_2 \in E$,alors $Sum\ e_1, e_2 \in E$
- si $e_1, e_2 \in E$,alors $Product\ e_1, e_2 \in E$

Cesontlesseulesfaçonsdeconstruireunélémentde E.

Exemples

- $Int\ 42$
- $Product\ Sum\ Int\ 2\ , Int\ 49\ , Int\ 5$

Remarques

- Attention : il s'agit d'une notation mathématique pour des *arbres*. Ce n'est pas une définition d'une syntaxe concrète.
- On utilise parfois aussi pour la définition des arbres de syntaxe abstraite un formalisme appelé *grammaire d'arbres*, qui risque créer une confusion. Pour nous, une grammaire définit toujours une syntaxe concrète.
- L'intérêt de cette notation est qu'elle nous permet de rédiger les règles de calcul sur la syntaxe abstraite. Cette notation est plus compacte que de dessiner un arbre.
- Souvent on définit plusieurs types d'arbre de syntaxe abstraite, par exemple un type pour les expressions arithmétiques, un autre pour les instructions d'un langage de programmation.

Traduction en JAVA

- Chaque type d'arbre correspond à une classe abstraite.
- Pour chacun des cas dans la définition il y a une classe dérivée.
- Le constructeur de cette classe dérivée a comme arguments les valeurs différentes utilisées dans la construction de l'arbre.
- Le constructeur stocke ces valeurs dans des champs privés de la classe.
- Éventuellement : des méthodes pour savoir dans quel cas on est, et pour récupérer les valeur utilisées dans la construction.

Définition de la Syntaxe Abstraite sur l'exemple I

```
abstract class Expression{
    public boolean isInt(){
        return false ;
    }
    public boolean IsSum(){
        return false ;
    }
    public boolean IsProduct(){
        return false ;
    }
}

class Int extends Expression{
    private int value;
```


Définition de la Syntaxe Abstraite sur l'exemple II

```
    public boolean isInt(){
        return true ;
    }
    public Int( int i){
        value=i;
    }
    public int getValue(){
        return value;
    }
}

class Sum extends Expression{
    private Expression left;
    private Expression right;
    public boolean isSum(){
```

DéfinitiondelaSyntaxeAbstraitesurl'exempleIII

```
            returntrue      ;
        }
        public   Sum(Expressione1,Expressione2){
            left=e1;
            right=e2;
        }
        public   Expression getLeft(){
            return   left;
        }
        public   Expression getRight(){
            return   right;
        }
}

class   Product   extends   Expression{
```


DéfinitiondelaSyntaxeAbstraitesurl'exempleIV

```
    private   Expressionleft;
    private   Expressionright;
    publicboolean   isProduct(){
        returntrue      ;
    }
    public   Product(Expressione1,Expressione2){
        left=e1;
        right=e2;
    }
    public   Expression getLeft(){
        return   left;
    }
    public   Expression getRight(){
        return   right;
    }
```


DéfinitiondelaSyntaxeAbstraitesurl'exempleV

```
}
```


LamêmechosedanslelangageOCaml

```
type   expression=
   |Integer   of   int
   |Sum   of   expression   expression
   |Product   of   expression   expression;;
```

(voirlecours *ProgrammationFonctionnelle* duL3)

Objectif

- Jusqu'à maintenant : syntaxe concrète et syntaxe abstraite
- Reste à faire :
  - Les restrictions supplémentaires qu'on ne peut pas exprimer par une grammaire : *analyse sémantique*
  - Exécution ou compilation d'un programme : *sémantique*
- Il nous faut d'abord une notation qui nous permet de spécifier précisément les restrictions supplémentaire, et la sémantique.
- Puis : codage (en Java, par exemple)

Langages de programmation : Trois étapes

1. De la syntaxe concrète à la syntaxe abstraite.
2. Analyse sémantique : par exemple assurer que toutes les variables sont déclarées, toutes les méthodes définies, que le typage est correct.
3. Exécution de la syntaxe abstraite (interprétation), ou traduction vers un autre langage (compilation, ou transformation de code)

L'enchaînement des trois étapes

1. Normalement : Construction de la syntaxe abstraite pour le programme entier avant toute analyse sémantique ou exécution (attention : il y des cas ou analyse syntaxique et exécution sont intercalées)
2. La vérification des conditions sémantiques est faite *avant* l'exécution. Pour cette raison on parle aussi parfois d'une sémantique *statique*.
3. L'exécution (ou la compilation) peut utiliser des résultats de l'étape (2). Dans ces cas, on l'étape (3) se base sur une syntaxe abstraite *annotée* (exemple : inférence de types).

Notion de jugement

- Un jugement est une affirmation portant sur une expression abstraite.
- Quelques exemples, en langue naturelle :
  - « l'expression *Sum Int* 1 , *Product Int* 2 , *Int* 3 a pour valeur 7 » ;
  - « l'expression *Sum Int* 1 , *Product Int* 2 , *Int* 3 a pour type int » ;
  - « l'instruction Write *Sum Int* 3 , *Int* 4 affiche 7 » ;
  - « l'expression *Sum Int* 3 , *Int* 5 a pour code machine [PUSHI 3; PUSHI 5; ADD] ».

Jugementsaveccontexte

- « dansle contexteoù x estunevariablevalant4, l'expression $Sum\ Int\ 1\ ,Product\ Var\ x\ ,Int\ 3$ apourvaleur13 »;
- « dansle contexteoù x estunevariabledetypereal, l'expression $Sum\ Int\ 1\ ,Product\ Var\ x\ ,Int\ 3$ apourtype real »;
- pourun compilateurdevantengendrerducodemachineàune adressedonnée, on pourra avoirdesjugementsdustyle « dans lecontexteoùl'adressecouranteducodeest250et x estune variabledontl'adresseest100, l'expression $Product\ Int\ 3\ ,Var\ x$ apourcodemachine[250PUSHI3; 251PUSH100;252MUL] ».

Définition

Un *jugement* estuntriplet $C, a, v \in D^c \times A \times D^v$ telque:

- A estunensembled'arbresdesyntaxeabstraite, contenant l'arbre a;
- D^v estunensembleappelé *domainedevaleurs* ; lavaleur v appartientausous-ensemble D^v desvaleurs;
- D^c estunensembleappelé *domainedecontextes* ; lavaleur C appartientausous-ensemble D^c desvaleurs.

Remarques

- Quandledomainedecontextesesttrivial, lejugementse réduitàunepaire a, v .
- Lescontextesetlesvaleurspeuventêtredespaires, triplets, ...

Notationetexemples

- Onnotelejugement $C, a, v \in D^c \times A \times D^v$ par « $C \vdash a \Rightarrow v$ ».
- Danslecasd'undomainedecontextetrivial: $a \Rightarrow v$.
- Exemples:
  - $Sum\ Int\ 1\ ,Product\ Int\ 2\ ,Int\ 3 \Rightarrow 7$

    estlejugement « l'expression1 2×3apourvaleur7 ».
  - $\{x \Rightarrow 4\} \vdash Sum\ Int\ 1\ ,Product\ Var\ x\ ,Int\ 3 \Rightarrow 13$

    estlejugement « dansle contexteoù x estunevariablevalant 4, l'expression1 $x \times 3$apourvaleur13 », ici D^v $\mathbb{N}$ et D^c estl'ensembledes *environnementsd'évaluation* .

Variantedenotation

Lorsqueledomainedevaleurs D^v estunensembledetypes, par exemple D^v $\{\text{int}, \text{bool}, \text{real}\}$, nousutiliseronsunevariante classiquedecettenotation, $C \vdash a$ t, où $t \in D^v$. Onécriradonc:

- $Sum\ Int\ 1\ ,Product\ Int\ 2\ ,Int\ 3$ int

  pourlejugement « l'expression1 2×3apourtypeint »;
- $\{x$ real$\} \vdash Sum\ Int\ 1\ ,Produit\ Var\ x\ ,Int\ 3$ real

  pourlejugement « dansle contexteoù x estunevariablede typereal, l'expression1 $x \times 3$apourtypereal ».

Jugementsvraisetfaux

Ladéfinitionprécédenteautorisel'écrituredejugementsvraisaussi bienquedejugementsfaux:

- «l'expression $Sum\ Int\ 1\ ,Mult\ Int\ 2\ ,Int\ 3$ apourvaleur 12»;
- «l'expression $Sum\ Int\ 1\ ,Mult\ Int\ 2\ ,Int\ 3$ apourtype bool»;

Axiomes

- Lesrèglessémantiquespermettentd'établirqu'uncertain jugementestvrai.
- Casdebase:Lesaxiomes:cesontdesjugementspour lesquelsonaffirmequ'ilssontvrais.
- Exemples(voirplustardpourlanotation):
  - si n estuneconstanteentière,alorslejugement $Int\ n \Rightarrow n$ est vrai;
  - si n estuneconstanteentière,alorslejugement $Int\ n\ \ int$ est vrai;
  - dansuncontexte C où x estunevariabledetypebool,le jugement $C \vdash Var\ x$ boolestvrai.

Lesrèglesengénéral

- Lesrèglespropermentditescorrespondentaucasgénéraldela récurrence.
- Ellesaffirmentlavéritéd'unjugementquiconstituela *conclusion* delarèglesousl'hypothèse(derécurrence)dela véritéd'unouplusieursjugementsappartenantaux *prémisses* delarègle.
- Parexemple:
  Silejugement $e_1\ \ int$ estvraietsilejugement $e_2\ \ int$ estvrai alorslejugement $Sum\ e_1, e_2\ \ int$ estégalementvrai.

Règlessémantiques

Définition

Une *règlesémantique* estun k-upletdejugements,avec $k \geq 1$.Si k 1,larègleestappeléeun *axiome*.Danslecasd'unerègle $J_1, \ldots, J_k$ avec $k \geq 2$, $J_1, \ldots,$ J_{k-1} constituentles *prémisses* de larègleet J_k enestla *conclusion*.

Notationhabituelle

Pourlarègle $J_1, \ldots, J_k$

$$\frac{J_1, \ldots, J_{k-1}}{J_k}$$

Danslecasd'unaxiome,riennefigureaudessusdutrait de fraction.

Lesschémasderègles

- Danslecasdel'évaluationd'uneexpressionarithmétiqueilya déjàuneinfinitéd'axiomes:

$$\frac{}{Int\ 0\ \Rightarrow 0} \qquad \frac{}{Int\ 1\ \Rightarrow 1} \quad \text{etc.}$$

- Pourcetteraison,onutilisedes *schémad'axiomes*

$$\text{pourtout } n \in \mathbb{N} : \quad \frac{}{Int\ n\ \Rightarrow n}$$

- Delamêmefaçon,onalesschémasderèglessuivantespour l'additionetlamultiplicationd'entiers:

$$\frac{e_1 \Rightarrow v_1 \qquad e_2 \Rightarrow v_2}{Sum\ e_1, e_2\ \Rightarrow v_1\ _{int}\ v_2}$$

et

$$\frac{e_1 \Rightarrow v_1 \qquad e_2 \Rightarrow v_2}{Product\ e_1, e_2\ \Rightarrow v_1 \times_{int} v_2}$$

Lesschémasderègles

- Dansunschémaderèglescomme

$$\frac{e_1 \Rightarrow v_1 \qquad e_2 \Rightarrow v_2}{Sum\ e_1, e_2\ \Rightarrow v_1\ _{int}\ v_2}$$

  e_1 et e_2 sontdes *méta-variables* qu'ilfautremplacerpardes arbresdesyntaxeabstraitepourobteniruneinstance.
- Souventonconfondlesnotionsderègleetdeschémaderègle.
- Lesopérateurs $_{int}$ et $\times_{int}$ sontlesopérationsarithmétiques.

Desrègleslsémantiqueàdesméthodesrécursives

- Dansdescassimples:lesjugementsspécifientunefonction, parexemplelafonctiond'évaluationquiassocieàchaquearbre desyntaxeabstraiteunevaleur *unique*.
- Cettefonctionpeutêtrepartielle.
- Dansnotrecasdel'évaluation:troisschémasderègles, chacunecorrespondantàuncasdansladéfinitiondesarbres desyntaxeabstraite.
- Ilyauneméthodeabstraite `ev l`.
- Chaquesous-classedéfinicetteméthodeselonleschéma donnée:
  - L'argumentdelaméthodeestlecontexte.
  - Lesprémissescorrespondentàdesappelsrécursifs.
  - Laconclusionditalorscommentcalculerlerésultat.

Évaluation:codel

```
abstract class Expression{
    abstract int eval();
}

class Int extends Expression{
    private int value;
    public Int( int i){
        value=i;
    }
    public int eval(){
        return value;
    }
}
```

Évaluation:codeII

```
class Sum extends Expression{
    private Expressionleft;
    private Expressionright;
    public Sum(Expressione1,Expressione2){
        left=e1;
        right=e2;
    }
    publicint eval(){
        return left.eval()+right.eval();
    }
}

class Product extends Expression{
    private Expressionleft;
    private Expressionright;
```


Évaluation:codeIII

```
    public Product(Expressione1,Expressione2){
        left=e1;
        right=e2;
    }
    publicint eval(){
        return left.eval() right.eval();
    }
}
```

AnalysedeDonnéesStructurées-Cours10

RalfTreinen

UniversitéParisDiderot
UFRInformatique
LaboratoirePreuves,ProgrammesetSystèmes

treinen@pps.univ-paris-diderot.fr

9avril2015

Rappel:évaluationd'expressionssansvariables

- L'ensemble E desarbresdesyntaxeabstraitereprésentantdes expressionsrégulièresestdéfinicommesuit:
  - tout $Int(n)$,où $n \in \mathbb{N}$,estunélémentde E
  - si $e_1, e_2 \in E$,alors $Sum(e_1, e_2) \in E$
  - si $e_1, e_2 \in E$,alors $Product(e_1, e_2) \in E$
- Règlessémantiques:

$$\frac{}{Int(n) \Rightarrow n} \quad n \in \mathbb{N}$$

$$\frac{e_1 \Rightarrow v_1 \qquad e_2 \Rightarrow v_2}{Sum(e_1, e_2) \Rightarrow v_1 +_{int} v_2}$$

$$\frac{e_1 \Rightarrow v_1 \qquad e_2 \Rightarrow v_2}{Product(e_1, e_2) \Rightarrow v_1 \times_{int} v_2}$$

Expressionsavecvariables

- L'ensemble E' desarbresdesyntaxeabstraitereprésentantdes expressionsrégulièresestdéfinicommesuit:
  - tout $Int(n)$,où $n \in \mathbb{N}$,estunélémentde E'
  - tout $Ident(s)$,où $s \in \Sigma^*$,estunélémentde E'
  - si $e_1, e_2 \in E'$,alors $Sum(e_1, e_2) \in E'$
  - si $e_1, e_2 \in E'$,alors $Product(e_1, e_2) \in E'$
- Σ estl'alphabetutilisépourformerdesnomsdevariables(par exemple,touslescaractèresUNICODE).
- Éventuellement,l'ensembledesnomsdevariablespossibles danslasyntaxeabstraiteestplusrichequelesnomsde variablespermisparlasyntaxeconcrète.

Évaluationd'expressionsavecvariables

- Maintenant,l'évaluationd'uneexpressiondépendaussid'un *contexte*.
- Ladéfinitionducontextedépenddel'application.Pour l'évaluation:Ils'agitd'unefonctionpartielle

$$\Sigma^* \to \mathbb{N}$$

aussiappelée *environnementdevaleurs* .
- Lejugementd'évaluation:

$$\Gamma \vdash e \Rightarrow v$$

exprime:Dansl'environnementdevaleur Γ,lavaleurde l'expression e est v.
- Exempled'unjugementvrai:

$$\{x = 3, y = 5\} \vdash Sum(Product(Ident(x), Int(2)), Ident(y)) \Rightarrow 11$$

Règlespourl'évaluationdesexpressionsavecvariables

- Axiomes:

$$\frac{}{\Gamma \vdash Int(n) \Rightarrow n} \quad n \in \mathbb{N}$$

$$\frac{}{\Gamma \vdash Ident(s) \Rightarrow n} \quad \Gamma(s) = n$$

- Règlesrécursives:

$$\frac{\Gamma \vdash e_1 \Rightarrow v_1 \qquad \Gamma \vdash e_2 \Rightarrow v_2}{\Gamma \vdash Sum(e_1, e_2) \Rightarrow v_1 +_{int} v_2}$$

$$\frac{\Gamma \vdash e_1 \Rightarrow v_1 \qquad \Gamma \vdash e_2 \Rightarrow v_2}{\Gamma \vdash Product(e_1, e_2) \Rightarrow v_1 \times_{int} v_2}$$

Remarques

- Quand $s \notin \text{domain}\ (\Gamma)$,alorsleschémad'axiomene s'appliquepasà $Ident(s)$.Danscecasonnepeutdoncpas calculerlavaleurd'uneexpressionquicontient $Ident(s)$.
- Danslesrèglesrécursives,comme

$$\frac{\Gamma \vdash e_1 \Rightarrow v_1 \qquad \Gamma \vdash e_2 \Rightarrow v_2}{\Gamma \vdash Sum(e_1, e_2) \Rightarrow v_1 +_{int} v_2}$$

lecontexte Γ utilisédansleshypothèsesestlemêmequedans laconclusion.Lecalculcorrespondantauxdeuxhypothèses peutdoncsefaireenparallèle.

Implémentation: `ValueEnvironment.java`

- Implémentationdesenvironnementsdevaleur:tablede hachage.

```
import java.util.HashMap;

class ValueEnvironment extends HashMap<String,Integer>{
    publicfinalstaticlong serialVersionUID=44;
}
```


Implémentation: `Expression.java` I

```
abstractclass Expression{
    abstractint eval(ValueEnvironmentenv);
}

class Int extends Expression{
    privateint value;
    public Int( int i){
        value=i;
    }
    publicint eval(ValueEnvironmentenv){
        return value;
    }
}
```

Implémentation: Expression.java II

```
class Ident extends Expression{
    private String name;
    public Ident(String s){
        name=s;
    }
    public int eval(ValueEnvironment env){
        return env.get(name);
    }
}

class Sum extends Expression{
    private Expression left;
    private Expression right;
    public Sum(Expression e1,Expression e2){
        left=e1;
```


Implémentation: Expression.java III

```
        right=e2;
    }
    public int eval(ValueEnvironment env){
        return left.eval(env)+right.eval(env);
    }
}

class Product extends Expression{
    private Expression left;
    private Expression right;
    public Product(Expression e1,Expression e2){
        left=e1;
        right=e2;
    }
    public int eval(ValueEnvironment env){
```


Implémentation: Expression.java IV

```
        return left.eval(env)          right.eval(env);
    }
}
```


AffectationetAffichage

- Objectif:exécutiondeprogrammes.
- Premièreétape:séquencesd'affectationàdesvariables,et d'instructionsd'affichage.
- Instruction"print"pourafficher,symbole":="pour l'affectation.
- Instructionsterminéesparlesymbole";"
- Fragmentdegrammaire(leresteestcommeavant):

$$
\begin{aligned}
S &\rightarrow \texttt{IL EOF} \\
\texttt{IL} &\rightarrow \epsilon \mid \texttt{I ; IL} \\
\texttt{I} &\rightarrow \texttt{ident := E} \mid \texttt{print E}
\end{aligned}
$$

Règlessémantiques

- Ons'intéressesurtoutàlamodificationdesenvironnements.
- Jugement $\Gamma_1 \vdash i \Rightarrow \Gamma_2$:l'exécutiondel'instruction(oudela listed'instructions) i dansl'environnement Γ_1 résultedans l'environnement Γ_2.
- Modélisationdesaffichages:omise.

Instructions

- Syntaxeabstraite:L'ensemble I desarbresdesyntaxe abstraitedesorte *Instruction* estdéfinicommesuit:
  - tout $Assign(s, e)$,où $s \in \Sigma^*$ et $e \in E'$,estunélémentde I ;
  - tout $Print(e)$,où $e \in E'$,estunélémentde I.
- Règlessémantiques:

$$\frac{\Gamma_1 \vdash e \Rightarrow v_1}{\Gamma_1 \vdash Affect(s, e) \Rightarrow \Gamma_1[s \mapsto v]}$$

$$\frac{}{\Gamma_1 \vdash Print(s, e) \Rightarrow \Gamma_1}$$

Remarques

- Maintenant,lecontextedansleshypothèsesn'estplus toujourslemêmequedanslaconclusion.
- Leshypothèsesdesrèglespeuventêtredesjugementsqui portentsurdesexpressionsdesyntaxeabstraitedesorte différentequelaconclusion(exemple:règlepour *Affect*).
- $\Gamma[s \mapsto v]$ estl'environnementquiassocieà s lavaleur v,etqui associe $\Gamma(x)$ àtout $x \neq s$.
- Ladeuxièmerègleexprimequel'exécutionde *Print* nemodifie pasl'environnement.

Listesd'instructions

- Syntaxeabstraite:L'ensemble IL desarbresdesyntaxe abstraitedesorte *InstructionList* estdéfinicommesuit:
  - Nil estunélémentde IL ;
  - Si $il \in IL$ et $i \in I$,alors $Seq(i, il)$ estunélémentde IL.
- Règlessémantiques:

$$\frac{}{\Gamma_1 \vdash Nil \Rightarrow \Gamma_1}$$

$$\frac{\Gamma_1 \vdash i \Rightarrow \Gamma_2 \qquad \Gamma_2 \vdash il \Rightarrow \Gamma_3}{\Gamma_1 \vdash Seq(i, il) \Rightarrow \Gamma_3}$$

Remarques

- La première règle (qui est un axiome) exprime que l'exécution de *Nil* ne modifie pas l'environnement.
- La deuxième règle impose maintenant un ordre sur l'exécution des deux composantes i et il :
  - Donnée initialement : Γ_1 et $Seq(i, il)$.
  - Il faut d'abord exécuter la première hypothèse qui donne, à partir de Γ_1 et de i, l'environnement Γ_2.
  - Puis on peut exécuter la deuxième hypothèse qui donne, à partir de Γ_2 et de il, l'environnement Γ_3.
  - L'implémentation utilise des tables de hachage modifiables, il faut bien respecter l'ordre !

Implémentation : `Instruction.java` I

```
abstract class Instruction{
    abstract void exec(ValueEnvironment env);
}

class Assignment extends Instruction{
    private Expression expression;
    private String variable;
    public Assignment(String v, Expression e){
        expression=e;
        variable=v;
    }
    public void exec(ValueEnvironment env){
        env.put(variable,expression.eval(env));
    }
```


Implémentation : `Instruction.java` II

```
}

class Print extends Instruction{
    private Expression expression;
    public Print(Expression e){
        expression=e;
    }
    public void exec(ValueEnvironment env){
        System.out.println(expression.eval(env));
    }
}
```


Implémentation : `InstructionList.java` I

```
abstract class InstructionList{
    abstract void exec(ValueEnvironment env);
}

class Nil extends InstructionList{
    public Nil(){
    }
    public void exec(ValueEnvironment env){
    }
}

class Seq extends InstructionList{
    private Instruction head;
    private InstructionList rest;
```

Implémentation: `InstructionList.java` II

```
    public Seq(Instruction i, InstructionList il){
        head=i;
        rest=il;
    }
    public void exec(ValueEnvironment env){
        head.exec(env);
        rest.exec(env);
    }
}
```


Expressionset Types

- Pourl'exécutiondesprogrammesavec, parexempledes instructionconditionnellesetdesboucles, nousavonsbesoin desexpressionsbooléennes.
- Onnepeutpasfairela distinctionentreexpressionsentièreset expressionsbooléennesparla grammaireà causedesvariables.

Typagestatiqueettypagedynamique

- Typagestatique:vérificationdestypesaprèsconstructionde lasyntaxeabstraite, et *avant* l'exécution.
- Avantage:Plusdesécurité, unefoisletypagevérifiéparle compilateuronsaitquedeserreursde *type* nepeutplusse produire(saufcertainslangagesdeprogrammationsqui permettentdecontournerletypage).
- Avantage:letypagepeutannoterlasyntaxeabstraitepardes typesdesexpressions, cequisimplifiel'exécution. Désambiguïsondesopérateursurchargés.
- Exemples:Java, Ocaml, ...

Typagestatiqueettypagedynamique

- Typagedynamique:vérificationdestypespendant *exécution* duprogramme, quandonappliqueunopérateurà desvaleurs.
- Avantage:plusdeflexibilité(mais:flexibilitéaussipossible danslecasstatique, parex.polymorphie).
- Avantage:programmesmoinsverbeux, cariln'estpas nécessairededéclarerlesvariablesavecleurtype(mais:le mêmeavantagepeutêtreobtenuparune *inférencedetype*).
- Exemples:Python, bash, Perl.

Unpetittourdequelqueslangagesdeprogrammation

- bash:analysesyntaxiqueàlavolée.Possibilitédefaire interpréterunechaînedecaractèrecommeunecommandede lashell(commande ev 1).
- Python:analysesyntaxiquecomplèteavantexécution,typage dynamique.
- Java:typagestatique,déclarationsexplicites
- Ocaml:typagestatique,inférencedetype.

Bash:analysesyntaxiqueàlavolée

```
#!/bin/bash
#lesdeuxpremieresligness ontexecuteesavant
#quel'erreurdesyntaxeestdetectee.

echo "hello"
echo "bonjour"
iffi
```


Bash:eval

```
#!/bin/bash
#evaluationd'unechaineconstruitependant
#l'executionduprogramme.

mot1="ecrire"
mot2="hello"
mot3="bonjour"

mot4=‘ sed -e's/crire/cho/'<<<$mot1‘

eval "$mot4␣$mot2␣$mot3"
```


Python:analysesyntaxiquecomplète

```
#!/usr/bin/python
#erreurdesyntaxedetecteeavantexecution

print "hello"
print "bonjour"
if fi
```

Python:typagedynamique

```
#!/usr/bin/python
#unevariablepeutprendredesvaleursde
#typedifferent.

x=42
print x
x=True
print x
```

AnalysedeDonnéesStructurées-Cours10
└Typage

Python:typagedynamique

```
#!/usr/bin/python

#erreurdetypageseulementdetecteequand
#labrancheelseestexecutee

import sys
read=sys.stdin.readline()
if read!='coocoo\n':
    print 17+21
else :
    print ('abc'&'def')
```

AnalysedeDonnéesStructurées-Cours10
└Typage

OCaml:inférencedetypes

```
(* letypedelafonctionfestinfere *)
letsommexy=x+y;;

(* erreurdetypage *)
somme"hello""bonjour";;
```

AnalysedeDonnéesStructurées-Cours10
└Typage

Typagestatiqueettypagedynamique

- Danslecasd'untypagestatique,destypessontinitialement pardesdéclarationsassociésàdes *identificateurs*.Puis,une inférencedetypepeutdéduireletypepourchaqueexpression, etvérifierquetouslesopérateurssontutilisésencohérence avecletypedesexpressions.
- Danslecasd'untypagedynamique,lestypessontassociés aux *valeurs* (éventuellementrésultatd'uncalcul).C'estau momentdel'applicationdel'opérateurqu'uneincompatibilité detypespeutêtredétectée.

AnalysedeDonnéesStructurées-Cours11

RalfTreinen

UniversitéParisDiderot
UFRInformatique
LaboratoirePreuves,ProgrammesetSystèmes

treinen@pps.univ-paris-diderot.fr

15avril2015

SyntaxeAbstraite

L'ensemble E'' desarbresdesyntaxeabstraitereprésentantdes expressionsarithmétiquesetbooléennesestdéfinicommesuit:

- tout $Int(n)$,où $n \in \mathbb{N}$,estunélémentde E''
- $True$ et $False$ sontdesélémentsde E''
- tout $Ident(s)$,où $s \in \Sigma^*$,estunélémentde E''
- si $e_1, e_2 \in E''$,alors $Plus(e_1, e_2) \in E''$
- si $e_1, e_2 \in E''$,alors $Mult(e_1, e_2) \in E''$
- si $e_1, e_2 \in E''$,alors $Greater(e_1, e_2) \in E''$
- si $e_1, e_2 \in E''$,alors $Equal(e_1, e_2) \in E''$
- si $e_1, e_2 \in E''$,alors $And(e_1, e_2) \in E''$
- si $e_1, e_2 \in E''$,alors $Or(e_1, e_2) \in E''$
- si $e \in E''$,alors $Not(e) \in E''$

Règlesdetypage

- Environnementdetypage:fonctionpartielle $\Sigma^* \to \{int, bool\}$
- Jugement $\Gamma \vdash e : t$:Dansl'environnement Γ,l'expression e a letype t.

$$\frac{}{\Gamma \vdash Int(n) : int} \quad n \in \mathbb{N}$$

$$\frac{}{\Gamma \vdash True : bool} \qquad \frac{}{\Gamma \vdash False : bool}$$

$$\frac{}{\Gamma \vdash Ident(s) : \Gamma(s)} \quad s \in \Sigma^*$$

Règlesdetypage

$$\frac{\Gamma \vdash e_1 : int \qquad \Gamma \vdash e_2 : int}{\Gamma \vdash Plus(e_1, e_2) : int}$$

$$\frac{\Gamma \vdash e_1 : int \qquad \Gamma \vdash e_2 : int}{\Gamma \vdash Mult(e_1, e_2) : int}$$

$$\frac{\Gamma \vdash e_1 : int \qquad \Gamma \vdash e_2 : int}{\Gamma \vdash Equal(e_1, e_2) : bool}$$

$$\frac{\Gamma \vdash e_1 : int \qquad \Gamma \vdash e_2 : int}{\Gamma \vdash Greater(e_1, e_2) : bool}$$

Règlesdetypage

$$\frac{\Gamma \vdash e_1 : bool \qquad \Gamma \vdash e_2 : bool}{\Gamma \vdash And(e_1, e_2) : bool}$$

$$\frac{\Gamma \vdash e_1 : bool \qquad \Gamma \vdash e_2 : bool}{\Gamma \vdash Or(e_1, e_2) : bool}$$

$$\frac{\Gamma \vdash e : bool}{\Gamma \vdash Not(e) : bool}$$

Implémentation: Expression.java (début)I

```
abstract class Expression{
    Type type;
    final public Type getType(){
        return this.type;
    }
    abstract void setType(TypeEnvironment env) throws TypeException;
}

class Int extends Expression{
    private int value;
    public Int(int i){
        value=i;
    }
    public void setType(TypeEnvironment env){
```


Implémentation: Expression.java (début)II

```
        this.type=Type.Int;
    }
}

class True extends Expression{
    public True(){
    }
    public void setType(TypeEnvironment env){
        this.type=Type.Bool;
    }
}

class False extends Expression{
    public False(){
    }
```


Implémentation: Expression.java (début)III

```
    public void setType(TypeEnvironment env){
        this.type=Type.Bool;
    }
}

class Ident extends Expression{
    private String name;
    public Ident(String s){
        name=s;
    }
    public void setType(TypeEnvironment env){
        this.type=env.get(name);
    }
}
```

Implémentation: `Expression.java` (début)IV

```
class Sum extends Expression{
    private Expressionleft;
    private Expressionright;
    public Sum(Expressione1,Expressione2){
        left=e1;
        right=e2;
    }
    publicvoid setType(TypeEnvironmentenv) throws TypeException{
        left.setType(env);
        right.setType(env);
        if (left.getType()==Type.Int
            &&right.getType()==Type.Int){
            this .type=Type.Int;
        } else {
            thrownew TypeException("Sum");
```


Implémentation: `Expression.java` (début)V

```
            }
        }
    }
```


Unereprésentationplusgénérique?

- Celadevientassezfastidieuxetrépétitif.
- Imaginezqu'est-cequeçadonnerasionajouteencoreplus d'opérateurs,oudesnouvellessortes(flottants,chaînesde caractère, ...)
- Solutionpluscompacte,etplusfacileàgénéraliser:
  - Définirunesorted'opérateurs(uneparnombred'arguments)
  - Définirunopérateurd'applicationd'unopérateuraubon nombred'expressions.

Syntaxeabstraitegénérique:opérateursetconstantes

- L'ensemble O_2 desopérateursensyntaxeabstraiteest:

$$O_2 = \{Plus, Mult, Greater, Equal, And, Or\}$$

- L'ensemble O_1 desopérateursunairesensyntaxeabstraiteest

$$O_1 = \{Not\}$$

- L'ensemble O_0 desconstantesensyntaxeabstraiteest

$$O_0 = \{True, False\} \cup \{Int(n) \mid n \in \mathbb{N}\}$$

Syntaxeabstraite:expressions

L'ensemble E''' desexpressionsensyntaxeabstraiteest:

- $Ident(s) \in E'''$ pourtout $s \in \Sigma^*$
- si $o \in O_0$ alors $Const(o) \in E'''$ pourtout $n \in \mathbb{N}$
- si $o \in O_1$, $e \in E'''$ alors $Apply_1(o, e) \in E'''$
- si $o \in O_2$, $e_1, e_2 \in E'''$ alors $Apply_2(o, e_1, e_2) \in E'''$

AnalysedeDonnéesStructurées-Cours11
Typagestatique
Expressionsgénériques

Typagepourlaformegénérique

- L'ensembledestypesdebaseest $BT = \{int, bool\}$
- Onvadéfinirunefonction ot quiassocie:
  - àtouteconstante $o \in O_0$ unélémentde BT,
  - àtoutopérateurunaire $o \in O_1$ unepairedans $BT \times BT$
  - àtoutopérateurbinaire $o \in O_2$ untripletdans $BT \times BT \times BT$.
- Danstouslescas,lederniertypeestceluidurésultatde l'applicationdel'opérateur,etlestypesprécédentsceuxdes arguments.
- Parfoisnotation: $t_1 \times t_2 \to t_3$ aulieude (t_1, t_2, t_3).

AnalysedeDonnéesStructurées-Cours11
Typagestatique
Expressionsgénériques

Typagedesconstantesetopérateurs

- Constantes:

$$\begin{aligned} ot(True) = ot(False) &= bool \\ ot(Int(n)) &= int \end{aligned}$$

- Opérateursunaires:

$$ot(Not) = (bool, bool)$$

- Opérateursbinaires:

$$\begin{aligned} ot(Plus) = ot(Mult) &= (int, int, int) \\ ot(And) = ot(Or) &= (bool, bool, bool) \\ ot(Greater) = ot(Equal) &= (int, int, bool) \end{aligned}$$

AnalysedeDonnéesStructurées-Cours11
Typagestatique
Expressionsgénériques

Typagedesexpressions

$$\frac{}{\Gamma \vdash Ident(s) : \Gamma(s)} \qquad s \in \Sigma^*$$

$$\frac{}{\Gamma \vdash Const(o) : t} \qquad o \in O_0, ot(o) = t$$

$$\frac{\Gamma \vdash e_1 : t_1}{\Gamma \vdash Apply_1(o, e) : t} \qquad o \in O_1, ot(o) = (t_1, t)$$

$$\frac{\Gamma \vdash e_1 : t_1 \qquad \Gamma \vdash e_2 : t_2}{\Gamma \vdash Apply_2(o, e_1, e_2) : t} \qquad o \in O_2, ot(o) = (t_1, t_2, t)$$

SyntaxeAbstraite

- Définitionsmutuellementrécursivesdesinstructionsetdes Listesd'Instructions.
- L'ensemble I desInstructionsensyntaxeabstraiteest:
  - si $e \in E'''$ alors $Print(e) \in I$,
  - si $s \in \Sigma^*, e \in E'''$ alors $Assign(s, e) \in I$,
  - si $e \in E''', il_1, il_2 \in IL$ alors $Cond(e, il_1, il_2) \in I$,
  - si $e \in E''', il \in IL$ alors $While(e, il) \in I$.
- L'ensemble IL desListesd'Instructionsensyntaxeabstraite est:
  - $Nil \in IL$,
  - si $i \in I$ et $il \in IL$ alors $Seq(i, il) \in IL$.

AnalysedeDonnéesStructurées-Cours11
Typagestatique
InstructionsetProgrammes

SyntaxeAbstraite

- L'ensemble D desDéclarationensyntaxeabstraiteest:
  - si $s \in \Sigma^*$ et $t \in BT$ alors $Decl(s, t) \in D$.
- L'ensemble DL desListesdeDéclarationsensyntaxeabstraite est:
  - $DeclNil \in DL$,
  - si $d \in D$ et $dl \in DL$ alors $DeclSeq(d, dl) \in DL$.
- L'ensemble P desProgrammesensyntaxeabstraiteest:
  - si $dl \in DL$ et $il \in IL$ alors $Program(dl, il) \in P$.

AnalysedeDonnéesStructurées-Cours11
Typagestatique
InstructionsetProgrammes

Déclarationsettypage

- Jugement $dl : \Gamma$:lalistededéclaration dl donne l'environnementdetypage Γ.
- Règles:

$$\frac{}{DeclNil : \emptyset}$$

$$\frac{dl : \Gamma}{DeclSeq(Decl(s, t), dl) : \Gamma[s \mapsto t]} \quad \text{si } s \notin \text{domain } (\Gamma)$$

AnalysedeDonnéesStructurées-Cours11
Typagestatique
InstructionsetProgrammes

Implémentation: `DeclarationList.java` I

```
abstract class DeclarationList{
    abstract TypeEnvironment extract() throws DeclException;
}

class DeclNil extends DeclarationList{
    public DeclNil(){
    }
    public TypeEnvironment extract(){
        return new TypeEnvironment();
    }
}

class DeclSeq extends DeclarationList{
    private Declaration head;
```

Implémentation: DeclarationList.java II

```
private DeclarationList rest;
public DeclSeq(Declaration d, DeclarationList dl){
    head=d;
    rest=dl;
}
public TypeEnvironment extract() throws DeclException{
    TypeEnvironment env=rest.extract();
    String head_var=head.getVariable();
        if (env.containsKey(head_var)){
            throw new DeclException(head_var);
        } else {
            env.put(head_var,head.getType());
            return env;
        }
}
```


Implémentation: DeclarationList.java III

```
}
```


Typagedelistesinstructions

- Jugement $\Gamma \vdash il : il$ estcorrectementtypéedans Γ.
- Règles:

$$\frac{}{\Gamma \vdash Nil}$$

$$\frac{\Gamma \vdash i \qquad \Gamma \vdash il}{\Gamma \vdash Seq(i, il)}$$

Implémentation: InstructionList.java I

```
abstract class InstructionList{
    abstract void checkTypes(TypeEnvironment env) throws TypeExceptio
}

class Nil extends InstructionList{
    public Nil(){
    }
    public void checkTypes(TypeEnvironment env){
    }
}

class Seq extends InstructionList{
    private Instruction head;
    private InstructionList rest;
```

Implémentation: `InstructionList.java` II

```
    public Seq(Instruction i, InstructionList il){
        head=i;
        rest=il;
    }
    public void checkTypes(TypeEnvironment env) throws TypeException{
        head.checkTypes(env);
        rest.checkTypes(env);
    }
}
```


Typaged'instructions

- Jugement $\Gamma \vdash i : i$ estcorrectementtypéedans Γ.
- Règlepourl'affichage:

$$\frac{\Gamma \vdash e : int}{\Gamma \vdash Print(e)}$$

- Règlepourl'affectation:

$$\frac{\Gamma \vdash e : \Gamma(s)}{\Gamma \vdash Affect(s, e)}$$

Typaged'instructions

- Règlepourlaconditionnelle:

$$\frac{\Gamma \vdash e : bool \qquad \Gamma \vdash il_1 \qquad \Gamma \vdash il_2}{\Gamma \vdash Cond(e, il_1, il_2)}$$

- Règlepourleboucle:

$$\frac{\Gamma \vdash e : bool \qquad \Gamma \vdash il}{\Gamma \vdash While(e, il)}$$

Implémentation: `Instruction.java` I

```
abstract class Instruction{
    abstract void checkTypes(TypeEnvironment env) throws TypeExceptio
}

class Assignment extends Instruction{
    private Expression expression;
    private String variable;
    public Assignment(String v, Expression e){
        expression=e;
        variable=v;
    }
    public void checkTypes(TypeEnvironment env) throws TypeException{
        expression.setType(env);
        if (expression.getType()!=env.get(variable)){
```

Implémentation: Instruction.java II

```
                thrownew TypeException("Assignment to "+variable);
            }
        }
    }

    class Print extends Instruction{
        private Expressionexpression;
        public Print(Expressione){
            expression=e;
        }
        publicvoid checkTypes(TypeEnvironmentenv) throws TypeException{
            expression.setType(env);
            if (expression.getType()!=Type.Int){
                thrownew TypeException("Print");
            }
```


Implémentation: Instruction.java III

```
        }
    }

    class While extends Instruction{
        private Expressioncondition;
        private InstructionListbody;
        public While(Expressione,InstructionListil){
            condition=e;
            body=il;
        }
        publicvoid checkTypes(TypeEnvironmentenv) throws TypeException{
            condition.setType(env);
            if (condition.getType()!=Type.Bool){
                thrownew TypeException("While");
            } else {
```


Implémentation: Instruction.java IV

```
                body.checkTypes(env);
            }
        }
    }

    class Conditional extends Instruction{
        private Expressioncondition;
        private InstructionListpositiv;
        private InstructionListnegativ;
        public Conditional(Expressione,InstructionListil1,InstructionListil2){
            condition=e;
            positiv=il1;
            negativ=il2;
        }
        publicvoid checkTypes(TypeEnvironmentenv) throws TypeException{
```


Implémentation: Instruction.java V

```
            condition.setType(env);
            if (condition.getType()!=Type.Bool){
                thrownew TypeException("Conditional");
            } else {
                positiv.checkTypes(env);
                negativ.checkTypes(env);
            }
        }
    }
```

Deux approches

- Les valeurs obtenues de l'évaluation d'une expression peuvent être de type différent (ici: *bool* ou *int*).
- Première résolution : Construire un type *somme* qui réunit toutes les valeurs de tous les types.
- Deuxième résolution : Des environnements de valeurs et des méthodes d'évaluation différents pour les types différents.
- La deuxième résolution est plus simple tant qu'on n'a que deux types, la première résolution est plus générique.

Approche 2 : utiliser deux environnements

- Deux environnements :

$$\begin{aligned} \Gamma_i &: \Sigma^* \to \mathbb{N} \\ \Gamma_b &: \Sigma^* \to \{true, false\} \end{aligned}$$

- Les deux fonctions sont partielles.
- Le typage nous garantie que

$$\text{domain } (\Gamma_i) \cap \text{domain } (\Gamma_b) = \emptyset$$

Jugements pour l'évaluation et l'exécution

- Forme des jugements pour l'évaluation :

$$\Gamma_i, \Gamma_b \vdash e \Rightarrow_i v_i$$
$$\Gamma_i, \Gamma_b \vdash e \Rightarrow_b v_b$$

où $v_i \in \mathbb{N}$, $v_b \in \{true, false\}$.

- Jugements et règles pour le traitement des déclarations : omis.
- Forme des jugements pour les instructions et les listes d'instructions :

$$\Gamma_i, \Gamma_b \vdash i \Rightarrow \Gamma'_i, \Gamma'_b$$
$$\Gamma_i, \Gamma_b \vdash il \Rightarrow \Gamma'_i, \Gamma'_b$$

Règles : transparents suivants.

Règles pour l'évaluation des expressions

$$\frac{}{\Gamma_i, \Gamma_b \vdash Int(n) \to_i n} \quad n \in \mathbb{N}$$

$$\frac{}{\Gamma_i, \Gamma_b \vdash True \to_b true} \qquad \frac{}{\Gamma_i, \Gamma_b \vdash False \to_b false}$$

$$\frac{}{\Gamma_i, \Gamma_b \vdash Ident(s) \to_i \Gamma_i(s)} \quad s \in \Sigma^*, \text{si } s \in \text{domain } (\Gamma_i)$$

$$\frac{}{\Gamma_i, \Gamma_b \vdash Ident(s) \to_b \Gamma_b(s)} \quad s \in \Sigma^*, \text{si } s \in \text{domain } (\Gamma_b)$$

Règlespourl'évaluationdesexpressions

$$\frac{\Gamma_i, \Gamma_b \vdash e_1 \rightarrow_i v_1 \qquad \Gamma_i, \Gamma_b \vdash e_2 \rightarrow_i v_2}{\Gamma_i, \Gamma_b \vdash \mathit{Plus}(e_1, e_2) \rightarrow_i v_1 +_{\mathit{int}} v_2}$$

$$\frac{\Gamma_i, \Gamma_b \vdash e_1 \rightarrow_i v_1 \qquad \Gamma_i, \Gamma_b \vdash e_2 \rightarrow_i v_2}{\Gamma_i, \Gamma_b \vdash \mathit{Equal}(e_1, e_2) \rightarrow_b \mathit{true}} \quad \text{si } v_1 = v_2$$

$$\frac{\Gamma_i, \Gamma_b \vdash e_1 \rightarrow_i v_1 \qquad \Gamma_i, \Gamma_b \vdash e_2 \rightarrow_i v_2}{\Gamma_i, \Gamma_b \vdash \mathit{Equal}(e_1, e_2) \rightarrow_b \mathit{false}} \quad \text{si } v_1 \neq v_2$$

etpareilpourlesautresopérateurs.

Implémentation: Expression.java (début)I

```
abstractclass Expression{
    Typetype;
    finalpublic TypegetType(){
        returnthis .type;
    }
    abstractvoid setType(TypeEnvironmentenv) throws TypeException;
    //default,willberedefinedforintegervaluedexpressions
    publicint evalInt(IntEnvironmentienv,BoolEnvironmentbenv)
        throws EvalException{
        thrownew EvalException();
    }
    //default,willberedefinedforboolvaluedexpressions
    publicboolean evalBool(IntEnvironmentienv,BoolEnvironmentbenv)
        throws EvalException{
```


Implémentation: Expression.java (début)II

```
            thrownew EvalException();
        }
    }

    class Int extends Expression{
        privateint value;
        public Int( int i){
            value=i;
        }
        publicvoid setType(TypeEnvironmentenv){
            this .type=Type.Int;
        }
        publicint evalInt(IntEnvironmentienv,BoolEnvironmentbenv){
            return value;
        }
```


Implémentation: Expression.java (début)III

```
    }

    class True extends Expression{
        public True(){
        }
        publicvoid setType(TypeEnvironmentenv){
            this .type=Type.Bool;
        }
        publicboolean evalBool(IntEnvironmentienv,BoolEnvironmentbenv){
            returntrue ;
        }
    }

    class Ident extends Expression{
        private Stringname;
```

Implémentation: Expression.java (début)IV

```
        public Ident(Strings){
            name=s;
        }
        publicvoid setType(TypeEnvironmentenv){
            this.type=env.get(name);
        }
        publicint evalInt(IntEnvironmentienv,BoolEnvironmentbenv){
            return ienv.get(name);
        }
        publicboolean evalBool(IntEnvironmentienv,BoolEnvironmentbenv){
            return benv.get(name);
        }
    }

    class Sum extends Expression{
```


Implémentation: Expression.java (début)V

```
        private Expressionleft;
        private Expressionright;
        public Sum(Expressione1,Expressione2){
            left=e1;
            right=e2;
        }
        publicvoid setType(TypeEnvironmentenv) throws TypeException{
            left.setType(env);
            right.setType(env);
            if (left.getType()==Type.Int
                &&right.getType()==Type.Int){
                this.type=Type.Int;
            } else {
                thrownew TypeException("Sum");
            }
```


Implémentation: Expression.java (début)VI

```
        }
        publicint evalInt(IntEnvironmentienv,BoolEnvironmentbenv)
            throws EvalException{
            int value_left=left.evalInt(ienv,benv);
            int value_right=right.evalInt(ienv,benv);
            return value_left+value_right;
        }
    }

    class Equal extends Expression{
        private Expressionleft;
        private Expressionright;
        public Equal(Expressione1,Expressione2){
            left=e1;
            right=e2;
```


Implémentation: Expression.java (début)VII

```
        }
        publicvoid setType(TypeEnvironmentenv) throws TypeException{
            left.setType(env);
            right.setType(env);
            if (left.getType()==Type.Int
                &&right.getType()==Type.Int){
                this.type=Type.Bool;
            } else {
                thrownew TypeException("Equal");
            }
        }
        publicboolean evalBool(IntEnvironmentienv,BoolEnvironmentbenv)
            throws EvalException{
            int value_left=left.evalInt(ienv,benv);
            int value_right=right.evalInt(ienv,benv);
```

Implémentation: Expression.java (début)VIII

```
            return    value_left==value_right;
        }
    }
```


Règlesd'exécutiondelistesd'instructions

$$\frac{}{\Gamma_i, \Gamma_b \vdash Nil \Rightarrow \Gamma_i, \Gamma_b}$$

$$\frac{\Gamma_i, \Gamma_b \vdash i \Rightarrow \Gamma'_i, \Gamma'_b \qquad \Gamma'_i, \Gamma'_b \vdash il \Rightarrow \Gamma''_i, \Gamma''_b}{\Gamma_i, \Gamma_b \vdash Seq(i, il) \Rightarrow \Gamma''_i, \Gamma''_b}$$

Implémentation: InstructionList.java I

```
abstract class    InstructionList{
    abstract void    checkTypes(TypeEnvironment env)    throws    TypeException;
    abstract void    exec(IntEnvironment ienv, BoolEnvironment benv)
        throws    EvalException;
}

class    Nil    extends    InstructionList{
    public    Nil(){
    }
    public void    checkTypes(TypeEnvironment env){
    }
    public void    exec(IntEnvironment ienv, BoolEnvironment benv){
    }
}
```


Implémentation: InstructionList.java II

```
class    Seq    extends    InstructionList{
    private    Instruction head;
    private    InstructionList rest;
    public    Seq(Instruction i, InstructionList il){
        head=i;
        rest=il;
    }
    public void    checkTypes(TypeEnvironment env)    throws    TypeException{
        head.checkTypes(env);
        rest.checkTypes(env);
    }
    public void    exec(IntEnvironment ienv, BoolEnvironment benv)
        throws    EvalException{
        head.exec(ienv, benv);
```

Implémentation: `InstructionList.java` III

```
            rest.exec(ienv,benv);
        }
    }
```


Règlespourl'exécutiondesinstructions

- Règlepourl'affichage:

$$\frac{}{\Gamma_i, \Gamma_b \vdash \mathit{Print}(e) \Rightarrow \Gamma_i, \Gamma_b}$$

- Règlespourl'affectation:

$$\frac{\Gamma_i, \Gamma_b \vdash e \Rightarrow_i v}{\Gamma_i, \Gamma_b \vdash \mathit{Affect}(s, e) \Rightarrow \Gamma_i[s \mapsto v], \Gamma_b}$$

$$\frac{\Gamma_i, \Gamma_b \vdash e \Rightarrow_b v}{\Gamma_i, \Gamma_b \vdash \mathit{Affect}(s, e) \Rightarrow \Gamma_i, \Gamma_b[s \mapsto v]}$$

Règlespourl'exécutiondesinstructions

- Premièrerèglepourlaconditionnelle:

$$\frac{\Gamma_i, \Gamma_b \vdash e \Rightarrow_b \mathit{true} \qquad \Gamma_i, \Gamma_b \vdash il_1 \Rightarrow \Gamma'_i, \Gamma'_b}{\Gamma_i, \Gamma_b \vdash \mathit{Cond}(e, il_1, il_2) \Rightarrow \Gamma'_i, \Gamma'_b}$$

- Deuxièmerèglepourlaconditionnelle:

$$\frac{\Gamma_i, \Gamma_b \vdash e \Rightarrow_b \mathit{false} \qquad \Gamma_i, \Gamma_b \vdash il_2 \Rightarrow \Gamma'_i, \Gamma'_b}{\Gamma_i, \Gamma_b \vdash \mathit{Cond}(e, il_1, il_2) \Rightarrow \Gamma'_i, \Gamma'_b}$$

Règlespourl'exécutiondesinstructions

Règlespourlaboucle:

$$\frac{\Gamma_i, \Gamma_b \vdash e \Rightarrow_b \mathit{false}}{\Gamma_i, \Gamma_b \vdash \mathit{While}(e, il) \Rightarrow \Gamma_i, \Gamma_b}$$

$$\frac{\Gamma_i, \Gamma_b \vdash e \Rightarrow_b \mathit{true} \quad \Gamma_i, \Gamma_b \vdash il \Rightarrow \Gamma'_i, \Gamma'_b \quad \Gamma'_i, \Gamma'_b \vdash \mathit{While}(e, il) \Rightarrow \Gamma''_i, \Gamma''_b}{\Gamma_i, \Gamma_b \vdash \mathit{While}(e, il) \Rightarrow \Gamma''_i, \Gamma''_b}$$

Implémentation: Instruction.java I

```
abstract class Instruction{
    abstract void checkTypes(TypeEnvironment env) throws TypeException;
    abstract void exec(IntEnvironment ienv, BoolEnvironment benv)
        throws EvalException;
}

class Assignment extends Instruction{
    private Expression expression;
    private String variable;
    public Assignment(String v, Expression e){
        expression=e;
        variable=v;
    }
    public void checkTypes(TypeEnvironment env) throws TypeException{
```


Implémentation: Instruction.java II

```
            expression.setType(env);
            if (expression.getType()!=env.get(variable)){
                throw new TypeException("Assignment to "+variable);
            }
        }
        public void exec(IntEnvironment ienv, BoolEnvironment benv)
            throws EvalException{
            if (expression.getType()==Type.Int){
                ienv.put(variable, expression.evalInt(ienv,benv));
            } else {
                benv.put(variable, expression.evalBool(ienv,benv));
            }
        }
}
```


Implémentation: Instruction.java III

```
class Print extends Instruction{
    private Expression expression;
    public Print(Expression e){
        expression=e;
    }
    public void checkTypes(TypeEnvironment env) throws TypeException{
        expression.setType(env);
        if (expression.getType()!=Type.Int){
            throw new TypeException("Print");
        }
    }
    public void exec(IntEnvironment ienv, BoolEnvironment benv)
        throws EvalException{
        System.out.println(expression.evalInt(ienv,benv));
    }
```


Implémentation: Instruction.java IV

```
}

class Conditional extends Instruction{
    private Expression condition;
    private InstructionList positiv;
    private InstructionList negativ;
    public Conditional(Expression e, InstructionList il1, InstructionList il2){
        condition=e;
        positiv=il1;
        negativ=il2;
    }
    public void checkTypes(TypeEnvironment env) throws TypeException{
        condition.setType(env);
        if (condition.getType()!=Type.Bool){
            throw new TypeException("Conditional");
```

Implémentation: Instruction.java V

```
            } else {
                positiv.checkTypes(env);
                negativ.checkTypes(env);
            }
        }
        public void exec(IntEnvironment ienv, BoolEnvironment benv)
            throws EvalException{
            if (condition.evalBool(ienv,benv)){
                positiv.exec(ienv,benv);
            } else {
                negativ.exec(ienv,benv);
            }
        }
    }
```


Implémentation: Instruction.java VI

```
class While extends Instruction{
    private Expression condition;
    private InstructionList body;
    public While(Expression e, InstructionList il){
        condition=e;
        body=il;
    }
    public void checkTypes(TypeEnvironment env) throws TypeException{
        condition.setType(env);
        if (condition.getType()!=Type.Bool){
            throw new TypeException("While");
        } else {
            body.checkTypes(env);
        }
    }
```


Implémentation: Instruction.java VII

```
    public void exec(IntEnvironment ienv, BoolEnvironment benv)
        throws EvalException{
        if (condition.evalBool(ienv,benv)){
            body.exec(ienv,benv);
            this.exec(ienv,benv);
        }
    }
}
```

AnalysedeDonnéesStructurées-Cours12

RalfTreinen

UniversitéParisDiderot
UFRInformatique
LaboratoirePreuves,ProgrammesetSystèmes

treinen@pps.univ-paris-diderot.fr

6mai2015

Rappel

- Jusqu'àmaintenant:unpetitlangagedeprogrammation impératif
  - Deuxtypes: `integer` et `boolean`
  - Expressionsavecdesopérateursarithmétiquesetlogiques
  - Déclarationdesvariablesavecleurtypeaudébutdu programme
  - Instructionsd'affichageetd'affectation
  - Instructionscomposées:boucle `while...do...od`, conditionnelle `if...then...else...fi`

Exempled'unprogramme

```
var
   x:integer;
   y:integer;
   b:boolean;
   c:boolean;
begin
   x:=73;
   ifb&&c||!bthen
      print42+1    *x+y;
      y:=42;
   else
      y:=x+x;
   fi;
   while42>41doprint73;od;
end
```


Unlangageavecdesdéclarationlocales

- Extensiondulangage:déclarationslocales
- Onveuxmaintenantaussipermettredesdéclarationsde variablesavecportéelimitée.
- Pourcelailyaunedécisionfondamentaleàprendre:

  est-cequ'onveutpermettreuneredéclarationlocale d'unevariable?

Exemple:variablelocale

```
 1var
 2 x:integer;
 3 y:integer;
 4begin
 5 x:=73;
 6 y:=x+42;
 7 var
 8  z:integer;
 9 begin
10 z:=x+y;
11 print(z);
12 end;
13 print(x+y);
14end
```


Exemple:redéclarationd'unevariable

```
 1var
 2 x:integer;
 3 y:integer;
 4begin
 5 x:=73;
 6 y:=x+42;
 7 var
 8  x:boolean;
 9 begin
10 x:=y>17;
11 ifxthenprint(1);elseprint(2);fi;
12 end;
13 print(x+y);
14end
```


Java:redéclarationpasautorisée

```
/*  Erreurdecompilation          */
class   Scope{
      publicstaticvoid          main(String[]args){
            int   x=42;
            {
                  int   x=73;
            }
      }
}
```


Redéclarationdevariables

- EnJava:laredéclarationd'unevariabledansunblocn'estpas permise(erreurdecompilation).
- Ilyadesautreslangagesdeprogrammationquiadmettentdes redéclarationlocalesdevariable.
- Lesredéclarationsontaussipertinentespourl'implémentation deprocéduresquipeuventaccéderàdesvariablesglobales.
- Pourcetteraisonnousallonsétudierlesdeuxsolutions.

Extensiondelasyntaxeabstraite

- L'ensemble DL desListesdeDéclarationsest ...
- L'ensemble IL deslistesd'Instructionsest ...
- L'ensemble I desInstructionsensyntaxeabstraiteest:
  - si $e \in E'''$ alors $Print(e) \in I$,
  - si $s \in \ ^*, e \in E'''$ alors $Assign(s, e) \in I$,
  - si $e \in E''', il_1, il_2 \in IL$ alors $Cond(e, il_1, il_2) \in I$,
  - si $e \in E''', il \in IL$ alors $While(e, il) \in I$,
  - si $dl \in DL$ et $il \in IL$ alors $Bloc(dl, il) \in I$.

AnalysedeDonnéesStructurées-Cours12
└BloquesetPortées

Petitemodificationdelasémantique

- Jusqu'àmaintenant,unevariablen'apasreçuunevaleur initialeaumomentdeladéclaration.
- Modification:aumomentdeladéclaration,lavariablereçoit unevaleurinitiale(0pourlesentiers, $false$ pourlesbooléens).
- Raison:simplificationtechnique.
- Onauraitaussipuétendrelasyntaxeabstraiteparune expressiond'initialisationaumomentdeladéclarationd'une variable.

AnalysedeDonnéesStructurées-Cours12
└BloquesetPortées

Modification:exécutiondedéclarations

- Jugement $dl \Rightarrow \ _i, \ _b$:l'exécutiondelalistededéclarations dl créeunenvironnemententier $\ _i$,etunenvironnement booléen $\ _b$.
- Règles:

$$\frac{}{DeclNil \Rightarrow \emptyset, \emptyset}$$

$$\frac{dl \Rightarrow \ _i, \ _b}{DeclSeq(Decl(s, \texttt{int}), dl) \Rightarrow \ _i[s \mapsto 0], \ _b}$$

$$\frac{dl \Rightarrow \ _i, \ _b}{DeclSeq(Decl(s, \texttt{bool}), dl) \Rightarrow \ _i, \ _b[s \mapsto false]}$$

AnalysedeDonnéesStructurées-Cours12
└BloquesetPortées

Modification:exécutiond'unprogramme

- Jugement $\ _i, \ _b \vdash il \Rightarrow \ '_i, \ '_b$:l'exécutiondelaliste d'instructions il dansunenvironnementinitial $\ _i, \ _b$ donneun nouvelenvironnement $\ '_i, \ '_b$
- Règlemodifiéepourunprogrammecomplèt:

$$\frac{dl \Rightarrow \ _i, \ _b \qquad \ _i, \ _b \vdash il \Rightarrow \ '_i, \ '_b}{Prog(dl, il) \Rightarrow \ '_i, \ '_b}$$

Deux opérations sur les environnements

- Union disjointe : si $\ _1$ et $\ _2$ sont deux environnements (de typage ou de valeur) avec des *domaines disjoints*, alors $\ _1 \oplus \ _2$ est définie comme

$$(\ _1 \oplus \ _2)(x) = \begin{cases} \ _1(x) & \text{si } x \in \mathit{domaine}(\ _1) \\ \ _2(x) & \text{sinon} \end{cases}$$

- Restriction : si $\ $ est un environnement, et $D \subseteq \mathit{domaine}(\ )$, alors la restriction de $\ $ à D est définie comme

$$|_D\ (x) = \begin{cases} \ (x) & \text{si } x \in D \\ \text{non-défini} & \text{sinon} \end{cases}$$

Typage pour les blocs

- Rappel des jugements de typage :
  - $dl\ :\ $: la liste de déclarations $\ dl$ donne lieu à l'environnement de typage $\ $.
  - $\ \vdash il\ :$ la liste d'instructions $\ il$ est bien typée par rapport à l'environnement $\ $.
- Règle

$$\frac{dl \Rightarrow\ ' \qquad \oplus\ ' : il}{: \mathit{Bloc}(dl, il)}$$

Exécution des blocs

$$\frac{dl \Rightarrow\ '_i,\ '_b \qquad _i \oplus\ '_i,\ _b \oplus\ '_b \vdash il \Rightarrow\ ''_i,\ ''_b}{_i,\ _b \vdash \mathit{Bloc}(dl, il) \Rightarrow\ ''_i\ |_{\mathit{domaine}\ _i},\ ''_b\ |_{\mathit{domaine}\ _b}}$$

Une modélisation plus complexe

- La technique des extensions et restrictions des environnements ne fonctionne plus dans le cas des redéclarations de variable.
- Raison : Une redéclaration d'une variable $\ x$ masque, pour la durée d'exécution du bloc, une déclaration globale de $\ x$. Après avoir terminé le bloc, la variable $\ x$ globale devient de nouveau visible.
- Nous avons donc maintenant besoin d'une liste d'environnements : quand on entre dans un bloc, un nouvel environnement est ajouté à la fin de la liste ; quand on quitte un bloc, l'environnement le plus récent est supprimé.

Listesd'environnements

- Dansunelistedenvironnements,l'environnementleplus globalecetrouveàgauche,etl'environnementlepluslocalse trouveàdroite.
- Pourobtenirlavaleurd'unevariable,oupourlamodifier,on parcourtlalistededroiteàgauche:prioritéàladéclarationla pluslocale.
- Pournepastropalourdirlanotationnousignorons maintenantladistinctionentreenvironnemententieret environnementbooléen.

Analysededonnées Structurées-Cours12
└Blocsetportées
└Avecredéclarationslocales

Opérationssurleslistesd'environnements

- L'opération *get* permetderécupérerlavaleurd'unevariable:

$$get(s,(\ _1,\ldots,\ _n)) = \begin{cases} \ _n(s) & \text{si } s \in domaine(\ _n) \\ get(s,(\ _1,\ldots,\ _{n-1})) & \text{sinon} \end{cases}$$

- L'opération *set* permetdemodifierlavaleurd'unevariable:

$$set(s,v,(\ _1,\ldots,\ _n)) = \begin{cases} (\ _1,\ldots,\ _{n-1},\ _n[s \mapsto v]) & \text{si } s \in domaine(\ _n) \\ set(s,v,(\ _1,\ldots,\ _{n-1})),\ _n & \text{sinon} \end{cases}$$

Analysededonnées Structurées-Cours12
└Blocsetportées
└Avecredéclarationslocales

Règlesàmodifierpourleslistesd'environnements

- Évaluationd'unevariable

$$\frac{}{\ _1,\ldots,\ _n \vdash Ident(s) \rightarrow get(s,(\ _1,\ldots,\ _n))}$$

- Affectationd'unevariable

$$\frac{\ _1,\ldots,\ _n \vdash e \Rightarrow v}{\ _1,\ldots,\ _n \vdash Assign(s,e) \Rightarrow set(s,v,(\ _1,\ldots,\ _n))}$$

Analysededonnées Structurées-Cours12
└Blocsetportées
└Avecredéclarationslocales

Nouvellerèglepourl'exécutiond'unbloc

$$\frac{dl \Rightarrow\ _n\ _1 \qquad \ _1,\ldots,\ _n,\ _n\ _1 \vdash il \Rightarrow\ '_1,\ldots,\ '_n,\ '_n\ _1}{\ _1,\ldots,\ _n \vdash Bloc(dl,il) \Rightarrow\ '_1,\ldots,\ '_n}$$

Extensiondelasyntaxe

- Syntaxeconcrète:définitiond'uneprocédure(àunargument)
  par
  proc *nom*(*variable* : *type*)
  bloc
- Syntaxeabstraite:
  - définitiond'uneprocédure:nomdelaprocédure,nomdu paramètre,typeduparamètre,bloc.
  - listededéfinitionsdeprocédures: ...
  - bloc:listededéclarationsdevariables,listededéfinitionsde procédures,bloc.

Exemple:procédure

```
 1varx:integer;
 2procp(y:integer)
 3 varz:integer;
 4 begin
 5 z:=2          *y;
 6 printz;
 7 end
 8begin
 9 x:=5;
10 p(x);
11end
```


Exemple:procédureaccédantàunevariableglobale

```
 1varx:integer;
 2procp(y:integer)
 3 varz:integer;
 4 begin
 5 z:=2          *y;
 6 x:=z;
 7 end
 8begin
 9 x:=5;
10 p(x);
11 printx;
12end
```


Exemple:liaisonstatiqueoudynamique

```
 1varx:integer;
 2procp(y:integer)
 3 begin
 4 x:=x+y;
 5 end
 6begin
 7 x:=1;
 8 varx:integer;
 9 begin
10 x:=2;
11 p(1);
12 printx;
13 end
14end
```

Liaisonstatiqueoudynamique

- Liaisonenquestion:laconnexionentrel'utilisationd'unnom (d'unevariable,parex.),etladéclarationdumêmenom.
- Problème:onpeutavoirplusieursdéclarationsdumêmenom, laquellechoisir?
- Liaisondynamique:laliaisonestfaitesuivantlaprioritéàla déclarationlaplusslocale *aumomentdel'exécution* .
- Liaisonstatique:laliaisonestfaitesuivantlaprioritéàla déclarationlaplusslocale *aumomentdeladéfinitiondela procédure*.
  Tousleslangagesdeprogrammationmodernessuiventcette politique.