Sémantique et Traduction des Langages

Majeure Sciences et Ingénierie du Logiciel

Marc Pantel

2020 – 2021

Organisation

- Cours : 10 séances – Marc Pantel
- TD : 8 séances – Marc Pantel/Neeraj Singh
- TP : 9 séances – Marc Pantel/Neeraj Singh
- Mini Projets (20%) en binôme : 2 séances de suivi + 1 test
  Finalisation travaux de TP sur langage fonctionnel miniML
  Finalisation travaux de TP sur langage impératif miniC
- Projet (40%) en quadrinôme : 5 séances de suivi + 1 test
  Extension du langage miniC avec technologies objets
- Examen (40%) : 1h30 avec documents
- Urgent : Constituer les quadrinômes et binômes associés
- Alternative 1 : Pas de confinement
  - Travaux Dirigées en présenciel
  - Travaux Pratiques à décider
- Alternative 2 : Confinement
  - Combinaison TD/TP à distance

Organisation

- Cours : 10 séances – Marc Pantel
- TD : 8 séances – Marc Pantel/Neeraj Singh
- TP : 9 séances – Marc Pantel/Neeraj Singh
- Mini Projets (20%) en binôme : 2 séances de suivi + 1 test
  Finalisation travaux de TP sur langage fonctionnel miniML
  Finalisation travaux de TP sur langage impératif miniC
- Projet (40%) en quadrinôme : 5 séances de suivi + 1 test
  Extension du langage miniC avec technologies objets
- Examen (40%) : 1h30 avec documents
- Urgent : Constituer les quadrinômes et binômes associés
- Alternative 1 : Pas de confinement
  - Travaux Dirigées en présenciel
  - Travaux Pratiques à décider
- Alternative 2 : Confinement
  - Combinaison TD/TP à distance

Organisation

- Cours : 10 séances – Marc Pantel
- TD : 8 séances – Marc Pantel/Neeraj Singh
- TP : 9 séances – Marc Pantel/Neeraj Singh
- Mini Projets (20%) en binôme : 2 séances de suivi + 1 test
  Finalisation travaux de TP sur langage fonctionnel miniML
  Finalisation travaux de TP sur langage impératif miniC
- Projet (40%) en quadrinôme : 5 séances de suivi + 1 test
  Extension du langage miniC avec technologies objets
- Examen (40%) : 1h30 avec documents
- Urgent : Constituer les quadrinômes et binômes associés
- Alternative 1 : Pas de confinement
  - Travaux Dirigées en présenciel
  - Travaux Pratiques à décider
- Alternative 2 : Confinement
  - Combinaison TD/TP à distance

Organisation

- Cours : 10 séances – Marc Pantel
- TD : 8 séances – Marc Pantel/Neeraj Singh
- TP : 9 séances – Marc Pantel/Neeraj Singh
- Mini Projets (20%) en binôme : 2 séances de suivi + 1 test
  Finalisation travaux de TP sur langage fonctionnel miniML
  Finalisation travaux de TP sur langage impératif miniC
- Projet (40%) en quadrinôme : 5 séances de suivi + 1 test
  Extension du langage miniC avec technologies objets
- Examen (40%) : 1h30 avec documents
- Urgent : Constituer les quadrinômes et binômes associés
- Alternative 1 : Pas de confinement
  - Travaux Dirigées en présenciel
  - Travaux Pratiques à décider
- Alternative 2 : Confinement
  - Combinaison TD/TP à distance

Organisation

- Cours : 10 séances – Marc Pantel
- TD : 8 séances – Marc Pantel/Neeraj Singh
- TP : 9 séances – Marc Pantel/Neeraj Singh
- Mini Projets (20%) en binôme : 2 séances de suivi + 1 test
  Finalisation travaux de TP sur langage fonctionnel miniML
  Finalisation travaux de TP sur langage impératif miniC
- Projet (40%) en quadrinôme : 5 séances de suivi + 1 test
  Extension du langage miniC avec technologies objets
- Examen (40%) : 1h30 avec documents
- Urgent : Constituer les quadrinômes et binômes associés
- Alternative 1 : Pas de confinement
  - Travaux Dirigées en présenciel
  - Travaux Pratiques à décider
- Alternative 2 : Confinement
  - Combinaison TD/TP à distance

Plan du cours

- Introduction
  - Rappels : Modélisation, Automates et Graphes, GLS
  - Architecture générale
  - Formes de sémantique
- Interprétation
  - Sémantique opérationnelle
  - Sémantique axiomatique
- Compilation
  - Table des Symboles, Arbre abstrait
  - Typage
  - Modèle mémoire, Génération de code
  - Sémantique translationnelle, dénotationnelle
- Vérification de correction

Plan du cours

- Introduction
  - Rappels : Modélisation, Automates et Graphes, GLS
  - Architecture générale
  - Formes de sémantique
- Interprétation
  - Sémantique opérationnelle
  - Sémantique axiomatique
- Compilation
  - Table des Symboles, Arbre abstrait
  - Typage
  - Modèle mémoire, Génération de code
  - Sémantique translationnelle, dénotationnelle
- Vérification de correction

Plan du cours

- Introduction
  - Rappels : Modélisation, Automates et Graphes, GLS
  - Architecture générale
  - Formes de sémantique
- Interprétation
  - Sémantique opérationnelle
  - Sémantique axiomatique
- Compilation
  - Table des Symboles, Arbre abstrait
  - Typage
  - Modèle mémoire, Génération de code
  - Sémantique translationnelle, dénotationnelle
- Vérification de correction

Plan du cours

- Introduction
  - Rappels : Modélisation, Automates et Graphes, GLS
  - Architecture générale
  - Formes de sémantique
- Interprétation
  - Sémantique opérationnelle
  - Sémantique axiomatique
- Compilation
  - Table des Symboles, Arbre abstrait
  - Typage
  - Modèle mémoire, Génération de code
  - Sémantique translationnelle, dénotationnelle
- Vérification de correction

Rappels

- Modélisation :
  - Structure algébrique des langages
  - Spécification des langages :
    - Expressions régulières,
    - Grammaire (règles de production, EBNF, Conway)
- Automates et Théorie des Langages
  - Automates, Automates à piles, Analyseur descendant récursif
  - Générateurs d'analyseurs lexicaux et syntaxiques
- Ingénierie Dirigée par les Modèles
  - Métamodèles :
    - Représentation abstraite du langage (MOF),
    - Règles de bonne formation (OCL)
  - Syntaxe concrète texte : Xtext

Rappels

- Modélisation :
  - Structure algébrique des langages
  - Spécification des langages :
    - Expressions régulières,
    - Grammaire (règles de production, EBNF, Conway)
- **Automates et Théorie des Langages**
  - **Automates, Automates à piles, Analyseur descendant récursif**
  - **Générateurs d'analyseurs lexicaux et syntaxiques**
- Ingénierie Dirigée par les Modèles
  - Métamodèles :
    - Représentation abstraite du langage (MOF),
    - Règles de bonne formation (OCL)
  - Syntaxe concrète texte : Xtext

Rappels

- Modélisation :
  - Structure algébrique des langages
  - Spécification des langages :
    - Expressions régulières,
    - Grammaire (règles de production, EBNF, Conway)
- Automates et Théorie des Langages
  - Automates, Automates à piles, Analyseur descendant récursif
  - Générateurs d'analyseurs lexicaux et syntaxiques
- Ingénierie Dirigée par les Modèles
  - Métamodèles :
    - Représentation abstraite du langage (MOF),
    - Règles de bonne formation (OCL)
  - Syntaxe concrète texte : Xtext

Principes essentiels

Communication = Echange d'informations

Besoins :
- Représenter les informations possibles
- Reconnaitre une information
- Exploiter une information

Organisation stratifiée : information structurée

Informatique : Science du traitement de l'information

Computer science : Science de la « machine à calculer »

Essentiel :
- Description et manipulation de l'information (langage),
- Traitement d'une information quelconque,
- Traitement d'une manipulation quelconque

D'où :
- Description formelle du langage
- Génération automatique des outils de manipulation

Références bibliographiques

- Hopcroft, Ullman, Introduction to automata theory, languages and computation, Addison-Wesley, 1979.
- Stern, Fondements mathématiques de l'informatique, McGraw-Hill, 1990.
- Carton, Langages formels, calculabilité et complexité, Vuibert, 2008.
- Aho, Sethi, Ullman, Compilateurs : Principes, Techniques et Outils, InterEditions, 1989.
- Fisher, Leblanc, Crafting a compiler in ADA/in C, Benjamin Cummings, 1991.
- Wilhem, Maurer, Les compilateurs : Théorie, construction, génération, Masson, 1994.
- Appel, Modern Compiler Implementation in Java/ML/C, Cambridge University Press, 1998.
- Winskel, The formal semantics of programming languages : An introduction, MIT Press, 1993.
- Lämmel, Software Languages : Syntax, Semantics and Metaprogramming, Springer (under review), 2017.

Exemple : fichier /etc/hosts

- Fichier tel qu'il est affiché :

```
#␣Ceci␣est␣un␣commentaire

127.0.0.1       →hal9000␣localhost

#␣En␣voici␣un␣autre

147.127.18.144       →phoenix.enseeiht.fr
```

- Informations brutes : caractères

```
0000000    #  sp   C   e   c   i  sp   e   s   t  sp   u   n  sp   c   o
0000020    m   m   e   n   t   a   i   r   e  nl  nl   1   2   7   .   0
0000040    .   0   .   1  ht   h   a   l   9   0   0   0  sp   l   o   c
0000060    a   l   h   o   s   t  nl  nl   #  sp   E   n  sp   v   o   i
0000100    c   i  sp   u   n  sp   a   u   t   r   e  nl  nl   1   4   7
0000120    .   1   2   7   .   1   8   .   1   4   4  ht   p   h   o   e
0000140    n   i   x   .   e   n   s   e   e   i   h   t   .   f   r  nl
0000160
```

Analyse lexicale

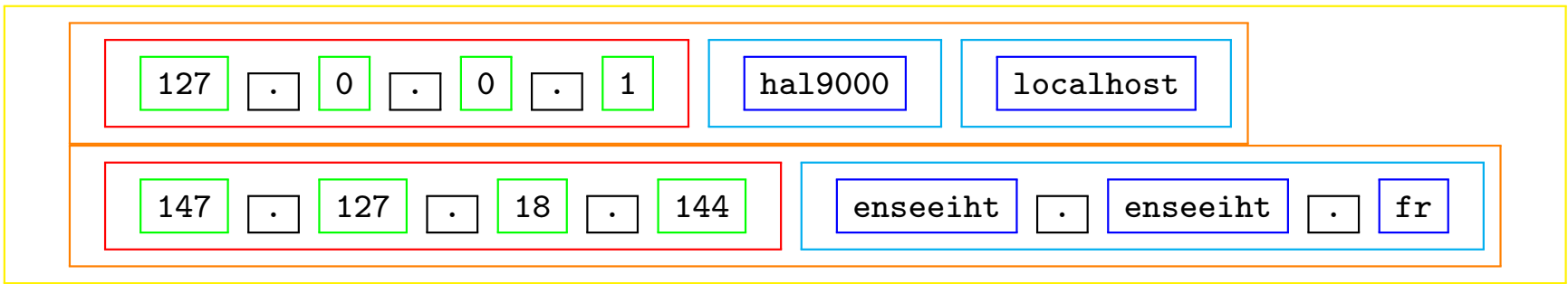
- Informations élémentaires : commentaire, nombre, identificateur, . (unités lexicales)
- Résultat de l'analyse lexicale :

`# Ceci est un commentaire` `127` `.` `0` `.` `0` `.` `1`
`hal9000` `localhost` `# En voici un autre` `147` `.`
`127` `.` `18` `.` `144` `enseeiht` `.` `enseeiht` `.` `fr`

- Spécification des unités lexicales : Expressions régulières
  - Commentaire : $\#[\hat{}\backslash n]^{\star}\backslash n$
  - Nombre : $[0-9]^{+}$
  - Identificateur : $[a-bA-B][a-bA-B0-9]^{\star}$

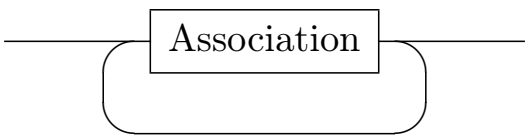
Analyse syntaxique

- Informations structurées (unités syntaxiques) :
  - Premier niveau : adresse IP, nom qualifié
  - Deuxième niveau : assocation
  - Troisième niveau : document

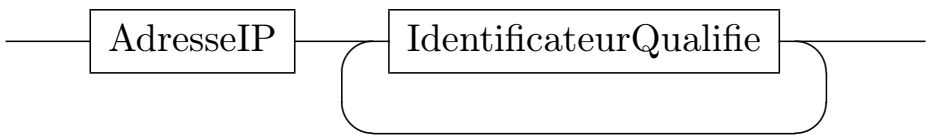


- Spécification des unités syntaxiques : Grammaires (notation de Conway)

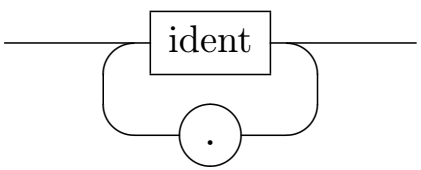
Document



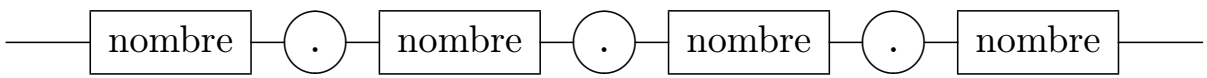
Association



IdentificateurQualifie

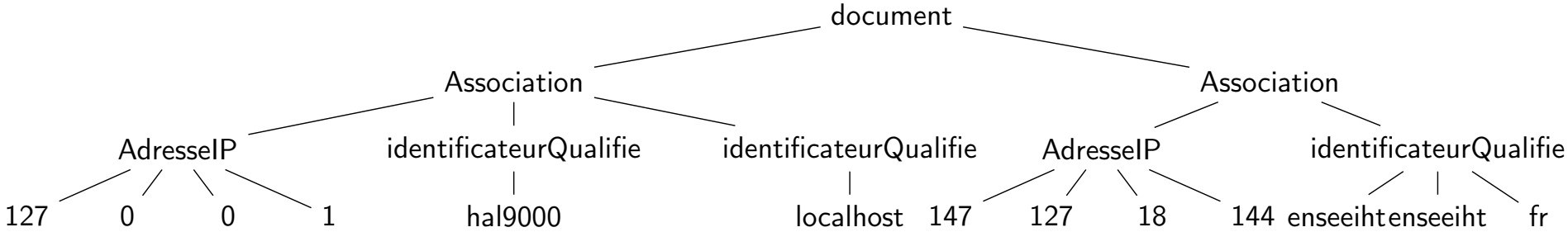


AdresseIP

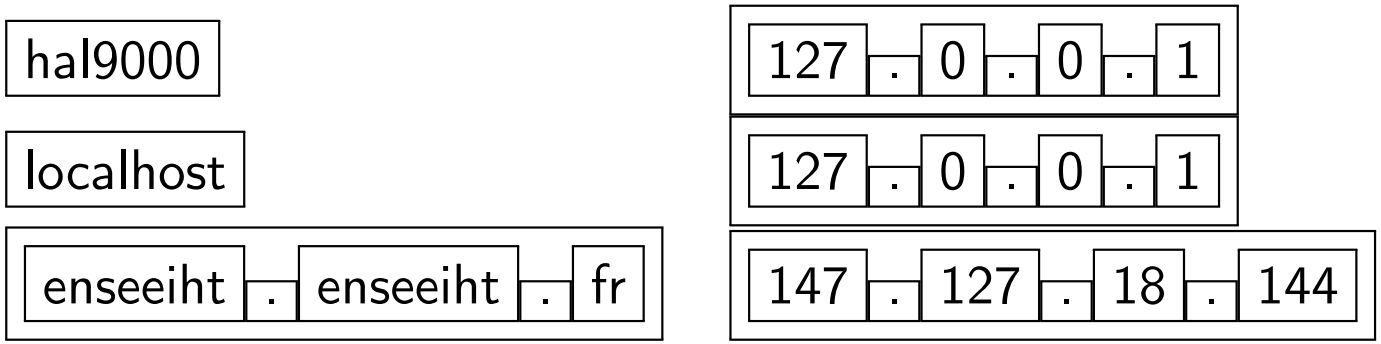


Analyse sémantique

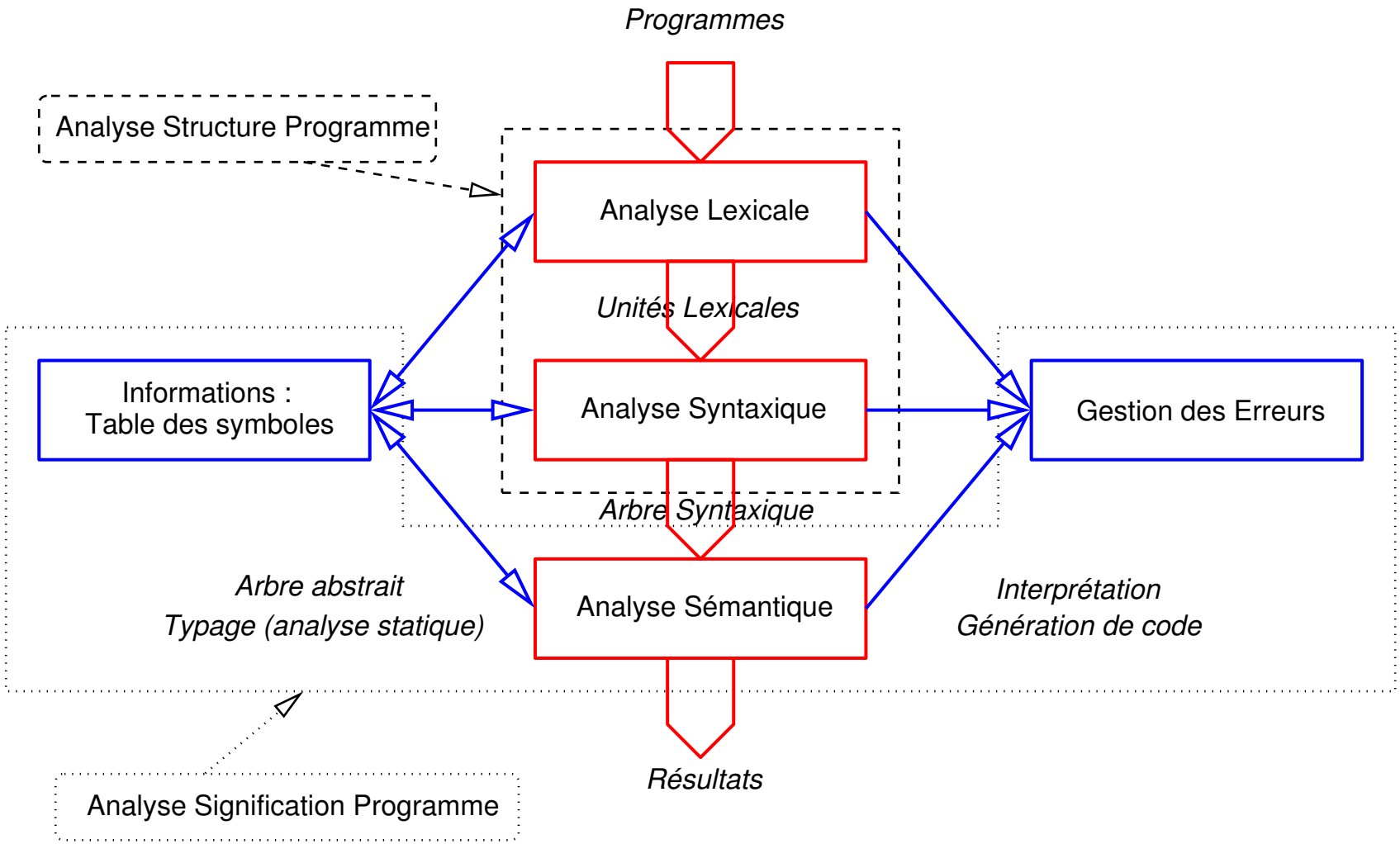
- Structure arborescente associée :



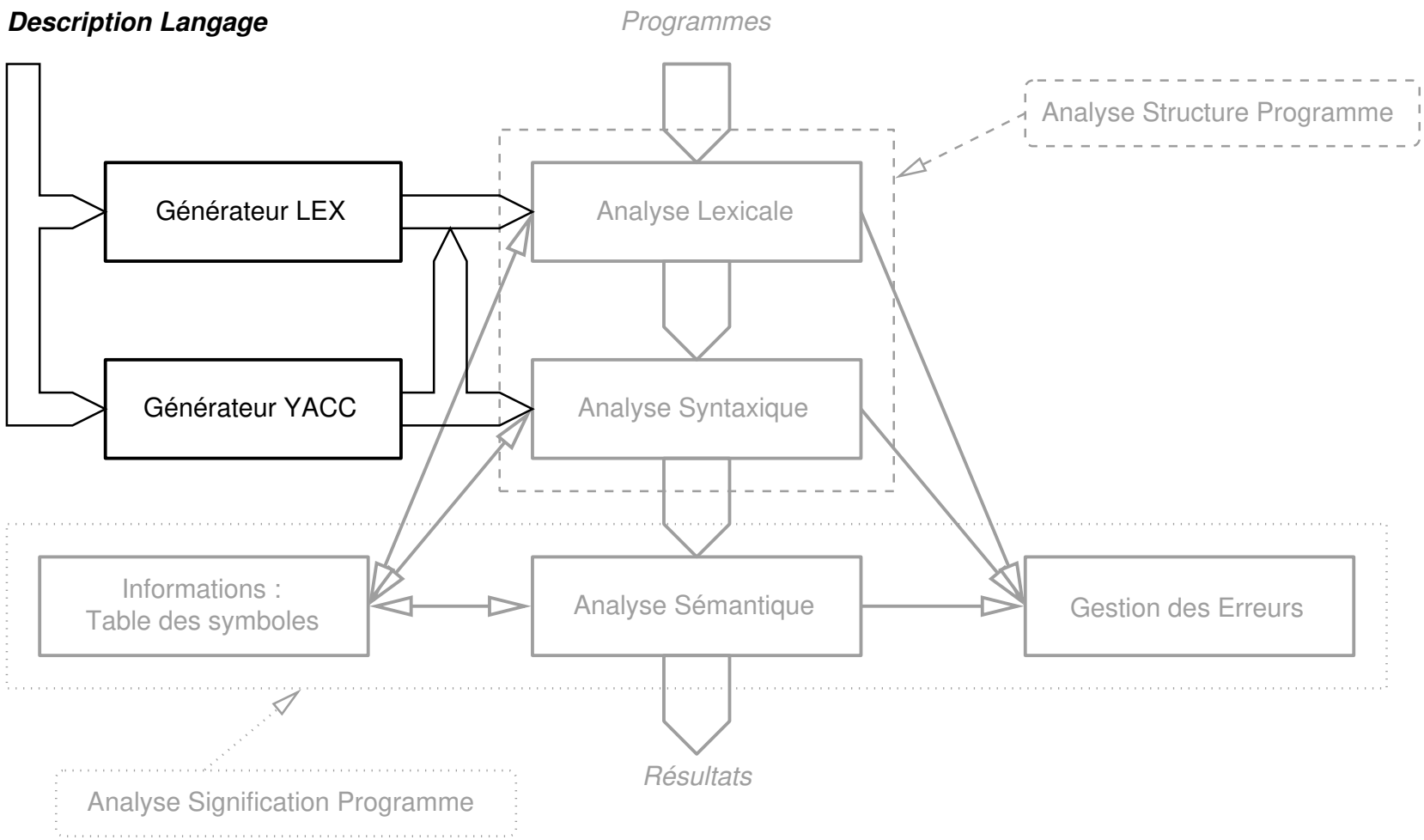
- Exploitation des informations : association nom qualifié/adresse IP (unités sémantiques)



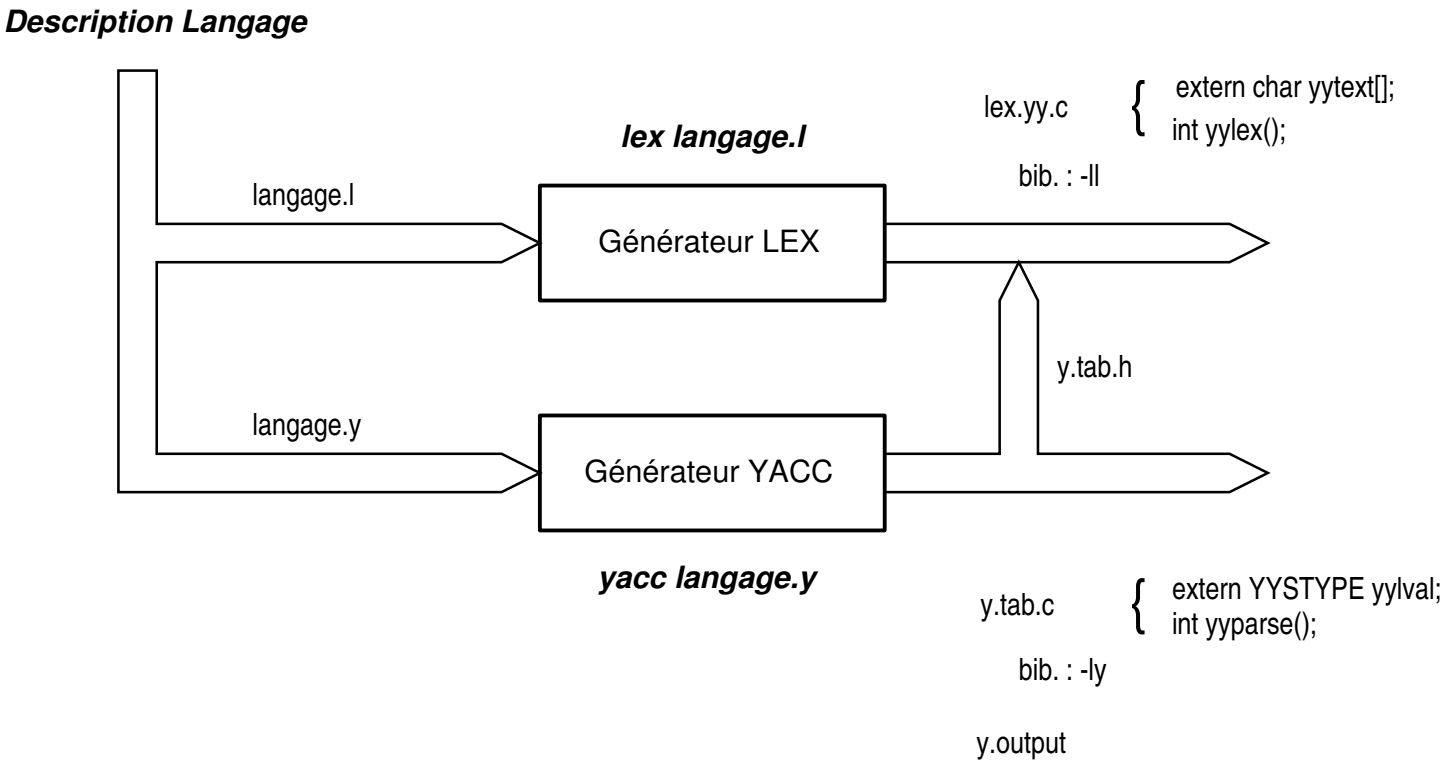
Structure d'un outil



Exemple lex et yacc



Exemple lex et yacc



Définitions

- Caractère/Symbole : Unité élémentaire d'information
- Unité lexicale (lexème, mot) : Séquence de caractères
- Unité syntaxique (arbre syntaxique, syntème, phrase) : Arbre d'unités lexicales
- Unité sémantique : diverses (arbre abstrait, table des symboles, type, code généré, résultat évaluation, ...)

Comment organiser les informations ?

- Objectif : Exploitation des informations
- Règle : Choisir le bon niveau de précision
- Unité lexicale : Bloc élémentaire d'information pertinente
- Unité syntaxique : Elément structurant de l'information

Sémantique formelle des langages

- Objectif : Modélisation la sémantique avec des outils mathématique
- Atteindre la qualité de la modélisation de la syntaxe
- Etudier la cohérence et la complétude
- Prouver la correction des outils
- Générer automatiquement les outils
- Différentes formes :
  - Sémantique opérationnelle : Mécanisme d'exécution des programmes
  - Sémantique axiomatique : Mécanisme de vérification des programmes
  - Sémantique translationnelle : Traduction vers un autre langage équipé d'une sémantique formelle
  - Sémantique dénotationnelle : Traduction vers un formalisme mathématique
- Validation des sémantique par étude équivalence entre formes

miniML

- Expressions : sans effets de bord, similaire dans tous les langages
- Partie fonctionnelle : sans effets de bord
- Partie Impérative : effets de bord, y compris dans les expressions et la partie fonctionnelle

miniML

- Expressions : sans effets de bord, similaire dans tous les langages
- Partie fonctionnelle : sans effets de bord
- Partie Impérative : effets de bord, y compris dans les expressions et la partie fonctionnelle

miniML

- Expressions : sans effets de bord, similaire dans tous les langages
- Partie fonctionnelle : sans effets de bord
- Partie Impérative : effets de bord, y compris dans les expressions et la partie fonctionnelle

miniML

- Expressions :

$$
\begin{array}{lcl}
\textit{Expr} & \rightarrow & \textit{Ident} \\
 & | & \textit{Const} \\
 & | & \textit{Expr Binaire Expr} \\
 & | & \textit{Unaire Expr} \\
 & | & (\ \textit{Expr}\ ) \\
 & & \\
\textit{Const} & \rightarrow & \textit{entier} \mid \textit{booleen} \\
 & & \\
\textit{Unaire} & \rightarrow & - \\
 & & \\
\textit{Binaire} & \rightarrow & + \mid - \mid * \mid / \mid \% \mid \& \mid \mid \\
 & | & == \mid != \mid < \mid <= \mid > \mid >=
\end{array}
$$

miniML

- Partie Fonctionnelle :

$$
\begin{array}{rcl}
\mathit{Expr} & \rightarrow & \cdots \\
& | & \texttt{let}\ \mathit{Ident}\ \texttt{=}\ \mathit{Expr}\ \texttt{in}\ \mathit{Expr} \\
& | & \texttt{if}\ \mathit{Expr}\ \texttt{then}\ \mathit{Expr}\ \texttt{else}\ \mathit{Expr} \\
& | & \texttt{fun}\ \mathit{Ident}\ \texttt{->}\ \mathit{Expr} \\
& | & \texttt{(}\mathit{Expr}\ \texttt{)}\ \mathit{Expr} \\
& | & \texttt{let rec}\ \mathit{Ident}\ \texttt{=}\ \mathit{Expr}\ \texttt{in}\ \mathit{Expr}
\end{array}
$$

- Partie Impératives :

$$
\begin{array}{rcl}
\mathit{Expr} & \rightarrow & \texttt{ref}\ \mathit{Expr} \\
& | & \texttt{!}\ \mathit{Expr} \\
& | & \mathit{Expr}\ \texttt{:=}\ \mathit{Expr} \\
& | & \mathit{Expr}\ \texttt{;}\ \mathit{Expr} \\
& | & \texttt{while}\ \mathit{Expr}\ \texttt{do}\ \mathit{Expr}\ \texttt{done}
\end{array}
$$

Interprétation : Principes généraux

- Programme qui exécute un programme (émulateur, machine virtuelle, ...)
- Langage hôte/support : Langage de programmation de l'interprète
- Représenter le programme comme une donnée
- Représenter l'exécution comme des données et algorithmes
  - Résultats de l'exécution (dont intermédiaires)
  - Ne pas oublier les erreurs d'exécution (résultats possibles)
  - Pour chaque construction exécutable du langage, identifier :
    1. Variantes selon résultats intermédiaires
    2. Étapes dans chaque variante
    3. Contraintes entre les étapes

Interprétation : Principes généraux

- Programme qui exécute un programme (émulateur, machine virtuelle, ...)
- Langage hôte/support : Langage de programmation de l'interprète
- Représenter le programme comme une donnée
- Représenter l'exécution comme des données et algorithmes
  - Résultats de l'exécution (dont intermédiaires)
  - Ne pas oublier les erreurs d'exécution (résultats possibles)
  - Pour chaque construction exécutable du langage, identifier :
    1. Variantes selon résultats intermédiaires
    2. Étapes dans chaque variante
    3. Contraintes entre les étapes

Interprétation : Principes généraux

- Programme qui exécute un programme (émulateur, machine virtuelle, ...)
- Langage hôte/support : Langage de programmation de l'interprète
- Représenter le programme comme une donnée
- Représenter l'exécution comme des données et algorithmes
  - Résultats de l'exécution (dont intermédiaires)
  - Ne pas oublier les erreurs d'exécution (résultats possibles)
  - Pour chaque construction exécutable du langage, identifier :
    1. Variantes selon résultats intermédiaires
    2. Étapes dans chaque variante
    3. Contraintes entre les étapes

Représentation des programmes et de l'exécution

- Programmes :
  - Arbres abstraits : Abstraction de l'arbre de dérivation (arbre syntaxique)
  - Structure de graphe (relation définition/utilisation)
    - Approche objet : Métamodèles
    - Approche fonctionnelle : Structure d'arbres + Tables des symboles
- Exécution :
  - Valeurs : Exploiter les types de base du langage hôte (booléen, entier, flottant, caractère, chaîne de caractère, ...)
  - Déclarations : Utilisation d'un dictionnaire (table des symboles)
  - Mémoire : Adresses et Espace de données associé

Représentation des programmes et de l'exécution

- Programmes :
  - Arbres abstraits : Abstraction de l'arbre de dérivation (arbre syntaxique)
  - Structure de graphe (relation définition/utilisation)
    - Approche objet : Métamodèles
    - Approche fonctionnelle : Structure d'arbres + Tables des symboles
- Exécution :
  - Valeurs : Exploiter les types de base du langage hôte (booléen, entier, flottant, caractère, chaîne de caractère, ...)
  - Déclarations : Utilisation d'un dictionnaire (table des symboles)
  - Mémoire : Adresses et Espace de données associé

Application à miniML

- Arbre abstrait : voir vidéo séparée
- Valeurs :

$$\begin{array}{lll} \mathit{Valeur} & \rightarrow & \mathit{Const} \\ & | & \bot \end{array}$$

- Algorithme d'exécution : voir vidéo séparée

Sémantique Opérationnelle

- Objectif : Décrire formellement les mécanismes d'exécution des programmes d'un langage
- Principe :
  - Exploiter la syntaxe du langage
  - Décrire l'exécution comme une transformation des programmes
- Notation : Règles de déduction
  - Soient $J_1, \ldots J_n$ et J des jugements :

	Notation	Signification
Déduction	$\dfrac{J_1 \quad J_n}{J}$	si J_1 et ...et J_n sont valides alors J est valide
Axiome	$\dfrac{}{J}$	J est valide

- Jugement d'exécution à grand pas : $\gamma \vdash e \Rightarrow v$
  - γ : environnement (association *Ident* / *Valeur*)
  - e : expression (*Expr*)
  - v : valeur (*Valeur*)
- Partie haute : Étapes intermédiaires (appels récursifs dans interpréte miniML)
- Partie basse : Construction traitée par la règle

miniML : Constantes et Accès identificateur

- Constante : Valeur ne change pas

$$\frac{}{\gamma \vdash \mathit{entier} \Rightarrow \mathit{entier}}$$

$$\frac{}{\gamma \vdash \mathit{booleen} \Rightarrow \mathit{booleen}}$$

- Identificateur : Accès à l'environnement
  - Présent : Transmission valeur associée

$$\frac{x \in \gamma \quad \gamma(x) = v}{\gamma \vdash x \Rightarrow v}$$

  - Absent : Cas d'erreur

$$\frac{x \notin \gamma}{\gamma \vdash x \Rightarrow \bot_{\mathit{undef}}}$$

miniML : Opérateur Unaire

- Étape préliminaire : Calcul du paramètre
- Variante 1 : Résultat correct du bon type

$$\frac{\gamma \vdash e \Rightarrow v \quad v \neq \bot \quad v \in dom\, op \quad v' = op\, v}{\gamma \vdash op\ e \Rightarrow v'}$$

- Variante 2 : Résultat erroné

$$\frac{\gamma \vdash e \Rightarrow v \quad v = \bot_c}{\gamma \vdash op\ e \Rightarrow \bot_c}$$

- Variante 3 : Résultat correct du mauvais type

$$\frac{\gamma \vdash e \Rightarrow v \quad v \neq \bot \quad v \notin dom\, op}{\gamma \vdash op\ e \Rightarrow \bot_{type}}$$

miniML : Opérateur Binaire

- Étapes préliminaires : Calcul des paramètres
- Question : Y a t'il un ordre particulier ?
- En absence d'effets de bord : Non, concurrence/parallélisme possible
- Variante 1 : Résultats corrects du bon type

$$\frac{\begin{array}{ll}\gamma \vdash e_1 \Rightarrow v_1 & v_1 \neq \bot \\ \gamma \vdash e_2 \Rightarrow v_2 & v_2 \neq \bot\end{array} \quad v_1 \times v_2 \in dom\, op \quad v = v_1\ op\ v_2}{\gamma \vdash e_1\ op\ e_2 \Rightarrow v}$$

- Variante 2 : Résultat(s) erroné(s)

$$\frac{\gamma \vdash e_1 \Rightarrow v_1 \quad v_1 = \bot_c}{\gamma \vdash e_1\ op\ e_2 \Rightarrow \bot_c} \qquad \frac{\gamma \vdash e_2 \Rightarrow v_2 \quad v_2 = \bot_c}{\gamma \vdash e_1\ op\ e_2 \Rightarrow \bot_c}$$

  - Que se passe t'il si deux erreurs se produisent de natures différentes ?
  - Définir une règle qui explicite ce cas

- Variante 3 : Résultat correct du mauvais type

$$\frac{\gamma \vdash e_1 \Rightarrow v_1 \quad \gamma \vdash e_2 \Rightarrow v_2 \quad v_1 \neq \bot \quad v_2 \neq \bot \quad v_1 \times v_2 \notin dom\, op}{\gamma \vdash e_1\ op\ e_2 \Rightarrow \bot_{type}}$$

miniML : Opérateur Binaire Droite à Gauche

- Imposons un ordre d'évaluation de droite à gauche (celui de OCaML)
- Variante 1 : Résultats corrects du bon type

$$\frac{\gamma \vdash e_2 \Rightarrow v_2 \quad v_2 \neq \bot \qquad \gamma \vdash e_1 \Rightarrow v_1 \quad v_1 \neq \bot \qquad v_1 \times v_2 \in dom\, op \quad v = v_1\ op\ v_2}{\gamma \vdash e_1\ op\ e_2 \Rightarrow v}$$

Attention : Cette règle n'impose pas d'ordre

- Variante 2 : Résultat(s) erroné(s)

$$\frac{\gamma \vdash e_2 \Rightarrow v_2 \quad v_2 = \bot_c}{\gamma \vdash e_1\ op\ e_2 \Rightarrow \bot_c} \qquad \frac{\gamma \vdash e_2 \Rightarrow v_2 \quad v_2 \neq \bot \qquad \gamma \vdash e_1 \Rightarrow v_1 \quad v_1 = \bot_c}{\gamma \vdash e_1\ op\ e_2 \Rightarrow \bot_c}$$

  - 2 erreurs ne peuvent plus se produire en même temps

- Variante 3 : Résultat correct du mauvais type

$$\frac{\gamma \vdash e_1 \Rightarrow v_1 \quad \gamma \vdash e_2 \Rightarrow v_2 \quad v_1 \neq \bot \quad v_2 \neq \bot \quad v_1 \times v_2 \notin dom\, op}{\gamma \vdash e_1\ op\ e_2 \Rightarrow \bot_{type}}$$

Exemple d'exécution d'un programme miniML

- Prenons : $\gamma = \{v \mapsto 2\}$
- Calculons le programme miniML : $1 + v * 3$
- L'arbre est trop volumineux, décomposons en :

$$A = \frac{v \in \gamma \quad \gamma(v) = 2}{\gamma \vdash v \Rightarrow 2}$$

$$B = \frac{\begin{array}{ccc} A & \gamma \vdash 3 \Rightarrow 3 & \\ 2 \neq \bot & 3 \neq \bot & 2 \times 3 \in dom* \quad 6 = 2 * 3 \end{array}}{\gamma \vdash v * 3 \Rightarrow 6}$$

$$\frac{\begin{array}{ccc} \gamma \vdash 1 \Rightarrow 1 & B & \\ 1 \neq \bot & 6 \neq \bot & 1 \times 6 \in dom+ \quad 7 = 1 + 6 \end{array}}{\gamma \vdash 1 + v * 3 \Rightarrow 7}$$

Définitions récursives

- Syntaxe : $\texttt{let rec } f \texttt{ = } e_1 \texttt{ in } e_2$
- Rappel : Définition simple

$$\frac{\gamma \vdash e_1 \Rightarrow v_1 \quad \gamma :: \{x \mapsto v_1\} \vdash e_2 \Rightarrow v_2}{\gamma \vdash \texttt{let } x \texttt{ = } e_1 \texttt{ in } e_2 \Rightarrow v_2}$$

- Rendons f visible dans e_1 :

$$\frac{\gamma :: \{f \mapsto v_1\} \vdash e_1 \Rightarrow v_1 \quad \gamma :: \{f \mapsto v_1\} \vdash e_2 \Rightarrow v_2}{\gamma \vdash \texttt{let rec } f \texttt{ = } e_1 \texttt{ in } e_2 \Rightarrow v_2}$$

- Question : Est ce bien fondé ?
- Remarque :
  $\texttt{let rec } f \texttt{ = } e_1 \texttt{ in } e_2 \equiv \texttt{let } f \texttt{ = let rec } f \texttt{ = } e_1 \texttt{ in } e_1 \texttt{ in } e_2$
- Exploitons cette relation :

$$\frac{\gamma \vdash \texttt{let rec } f \texttt{ = } e_1 \texttt{ in } e_1 \Rightarrow v_1 \quad \gamma :: \{f \mapsto v_1\} \vdash e_2 \Rightarrow v_2}{\gamma \vdash \texttt{let rec } f \texttt{ = } e_1 \texttt{ in } e_2 \Rightarrow v_2}$$

- Est ce un progrès ?

Définitions récursives

- Syntaxe : $\texttt{let rec } f = e_1 \texttt{ in } e_2$
- Rappel : Définition simple

$$\frac{\gamma \vdash e_1 \Rightarrow v_1 \quad \gamma :: \{x \mapsto v_1\} \vdash e_2 \Rightarrow v_2}{\gamma \vdash \texttt{let } x = e_1 \texttt{ in } e_2 \Rightarrow v_2}$$

- Rendons f visible dans e_1 :

$$\frac{\gamma :: \{f \mapsto v_1\} \vdash e_1 \Rightarrow v_1 \quad \gamma :: \{f \mapsto v_1\} \vdash e_2 \Rightarrow v_2}{\gamma \vdash \texttt{let rec } f = e_1 \texttt{ in } e_2 \Rightarrow v_2}$$

- Question : Est ce bien fondé ?
- Remarque :
  $\texttt{let rec } f = e_1 \texttt{ in } e_2 \equiv \texttt{let } f = \texttt{let rec } f = e_1 \texttt{ in } e_1 \texttt{ in } e_2$
- Exploitons cette relation :

$$\frac{\gamma \vdash \texttt{let rec } f = e_1 \texttt{ in } e_1 \Rightarrow v_1 \quad \gamma :: \{f \mapsto v_1\} \vdash e_2 \Rightarrow v_2}{\gamma \vdash \texttt{let rec } f = e_1 \texttt{ in } e_2 \Rightarrow v_2}$$

- Est ce un progrès ?

Définitions récursives

- Syntaxe : $\texttt{let rec } f = e_1 \texttt{ in } e_2$
- Rappel : Définition simple

$$\frac{\gamma \vdash e_1 \Rightarrow v_1 \quad \gamma :: \{x \mapsto v_1\} \vdash e_2 \Rightarrow v_2}{\gamma \vdash \texttt{let } x = e_1 \texttt{ in } e_2 \Rightarrow v_2}$$

- Rendons f visible dans e_1 :

$$\frac{\gamma :: \{f \mapsto v_1\} \vdash e_1 \Rightarrow v_1 \quad \gamma :: \{f \mapsto v_1\} \vdash e_2 \Rightarrow v_2}{\gamma \vdash \texttt{let rec } f = e_1 \texttt{ in } e_2 \Rightarrow v_2}$$

- Question : Est ce bien fondé ?
- Remarque :
  $\texttt{let rec } f = e_1 \texttt{ in } e_2 \equiv \texttt{let } f = \texttt{let rec } f = e_1 \texttt{ in } e_1 \texttt{ in } e_2$
- Exploitons cette relation :

$$\frac{\gamma \vdash \texttt{let rec } f = e_1 \texttt{ in } e_1 \Rightarrow v_1 \quad \gamma :: \{f \mapsto v_1\} \vdash e_2 \Rightarrow v_2}{\gamma \vdash \texttt{let rec } f = e_1 \texttt{ in } e_2 \Rightarrow v_2}$$

- Est ce un progrès ?

Définitions récursives

- Si nous le faisons une seconde fois :

$$\frac{\gamma \vdash \texttt{let rec } f = e_1 \texttt{ in } e_1 \Rightarrow v_1 \quad \gamma :: \{f \mapsto v_1\} \vdash e_1 \Rightarrow v_1}{\gamma \vdash \texttt{let rec } f = e_1 \texttt{ in } e_1 \Rightarrow v_1}$$

- Si e_1 s'évalue en une fonction $\langle\, \texttt{fun } x \texttt{ -> } e_3 \,,\, \gamma_{def} \,\rangle$
- Nous pouvons alors poursuivre le calcul de e_2 en exploitant cette fermeture
- Nous en déduisons la règle simplifiée dans laquelle nous gelons le calcul de la définition récursive

$$\frac{\gamma :: \{f \mapsto \langle\, \texttt{let rec } f = e_1 \texttt{ in } e_1 \,,\, \gamma \,\rangle\} \vdash e_2 \Rightarrow v}{\gamma \vdash \texttt{let rec } f = e_1 \texttt{ in } e_2 \Rightarrow v}$$

- Il faut alors ajouter une règle qui degèle le calcul lors de l'accès à f dans l'environnement :

$$\frac{x \in \gamma \quad \gamma(x) = \langle\, e \,,\, \gamma_{def} \,\rangle \quad \gamma_{def} \vdash e \Rightarrow v}{\gamma \vdash x \Rightarrow v}$$

$$\frac{x \in \gamma \quad \gamma(x) = v \quad v \neq \langle\, e \,,\, \gamma_{def} \,\rangle}{\gamma \vdash x \Rightarrow v}$$

Définitions récursives

- Si nous le faisons une seconde fois :

$$\frac{\gamma \vdash \texttt{let rec } f = e_1 \texttt{ in } e_1 \Rightarrow v_1 \quad \gamma :: \{f \mapsto v_1\} \vdash e_1 \Rightarrow v_1}{\gamma \vdash \texttt{let rec } f = e_1 \texttt{ in } e_1 \Rightarrow v_1}$$

- Si e_1 s'évalue en une fonction $\langle\, \texttt{fun } x \texttt{ -> } e_3 \,,\, \gamma_{def} \,\rangle$
- Nous pouvons alors poursuivre le calcul de e_2 en exploitant cette fermeture
- Nous en déduisons la règle simplifiée dans laquelle nous gelons le calcul de la définition récursive

$$\frac{\gamma :: \{f \mapsto \langle\, \texttt{let rec } f = e_1 \texttt{ in } e_1 \,,\, \gamma \,\rangle\} \vdash e_2 \Rightarrow v}{\gamma \vdash \texttt{let rec } f = e_1 \texttt{ in } e_2 \Rightarrow v}$$

- Il faut alors ajouter une règle qui degèle le calcul lors de l'accès à f dans l'environnement :

$$\frac{x \in \gamma \quad \gamma(x) = \langle\, e \,,\, \gamma_{def} \,\rangle \quad \gamma_{def} \vdash e \Rightarrow v}{\gamma \vdash x \Rightarrow v}$$

$$\frac{x \in \gamma \quad \gamma(x) = v \quad v \neq \langle\, e \,,\, \gamma_{def} \,\rangle}{\gamma \vdash x \Rightarrow v}$$

Analyse de programmes

- Objectif : Déterminer les propriétés des programmes
- Analyse dynamique : Exécuter les programmes pour observer les propriétés
- Approche incomplète :
  - Exécution finie : Nombre d'étapes d'exécution fini
  - Nombre d'exécution fini
- Analyse statique : Déterminer les propriétés sans exécuter les programmes
  - Abstraction finie d'une exécution
  - Exécution symbolique du programme (interprétation abstraite)
  - Approche complète : Abstraction de toutes étapes de toutes les exécutions possibles
  - Approche correcte : Sur-approximation des propriétés réelles
- Exemple : Détecter certaines erreurs d'exécution sans exécuter les programmes (Définitions, Typage, Erreurs de calcul, Consommation ressources, ...)

Analyse de programmes

- Objectif : Déterminer les propriétés des programmes
- Analyse dynamique : Exécuter les programmes pour observer les propriétés
- Approche incomplète :
  - Exécution finie : Nombre d'étapes d'exécution fini
  - Nombre d'exécution fini
- Analyse statique : Déterminer les propriétés sans exécuter les programmes
  - Abstraction finie d'une exécution
  - Exécution symbolique du programme (interprétation abstraite)
  - Approche complète : Abstraction de toutes étapes de toutes les exécutions possibles
  - Approche correcte : Sur-approximation des propriétés réelles
- Exemple : Détecter certaines erreurs d'exécution sans exécuter les programmes (Définitions, Typage, Erreurs de calcul, Consommation ressources, ...)

Analyse de programmes

- Objectif : Déterminer les propriétés des programmes
- Analyse dynamique : Exécuter les programmes pour observer les propriétés
- Approche incomplète :
  - Exécution finie : Nombre d'étapes d'exécution fini
  - Nombre d'exécution fini
- Analyse statique : Déterminer les propriétés sans exécuter les programmes
  - Abstraction finie d'une exécution
  - Exécution symbolique du programme (interprétation abstraite)
  - Approche complète : Abstraction de toutes étapes de toutes les exécutions possibles
  - Approche correcte : Sur-approximation des propriétés réelles
- Exemple : Détecter certaines erreurs d'exécution sans exécuter les programmes (Définitions, Typage, Erreurs de calcul, Consommation ressources, ...)

Analyse de programmes

- Objectif : Déterminer les propriétés des programmes
- Analyse dynamique : Exécuter les programmes pour observer les propriétés
- Approche incomplète :
  - Exécution finie : Nombre d'étapes d'exécution fini
  - Nombre d'exécution fini
- Analyse statique : Déterminer les propriétés sans exécuter les programmes
  - Abstraction finie d'une exécution
  - Exécution symbolique du programme (interprétation abstraite)
  - Approche complète : Abstraction de toutes étapes de toutes les exécutions possibles
  - Approche correcte : Sur-approximation des propriétés réelles
- Exemple : Détecter certaines erreurs d'exécution sans exécuter les programmes (Définitions, Typage, Erreurs de calcul, Consommation ressources, ...)

Mécanisme de typage

- Notion de type : Ensemble de valeurs pour lesquelles le programme a le même comportement
- Langage des types possibles : Syntaxe des types
  - Pour les expressions de miniML : `bool` et `int`
- Sémantique des types : Ensemble des valeurs possibles y compris les erreurs à l'exécution
  - $[\![\texttt{bool}]\!] = \{\texttt{true}, \texttt{false}, \bot_{runtime}\}$
  - $[\![\texttt{int}]\!] = \mathbb{Z} \cup \{\bot_{runtime}\}$
- Relations de comparaison des types :
  - Égalité
  - Compatibilité :
    - Généricité/Instanciation, Polymorphisme paramètrique
    - Sous-typage, Polymorphisme d'héritage
    - ...
- Règles de calcul (en miniML, Unification des types)

Mécanisme de typage

- Notion de type : Ensemble de valeurs pour lesquelles le programme a le même comportement
- Langage des types possibles : Syntaxe des types
  - Pour les expressions de miniML : `bool` et `int`
- Sémantique des types : Ensemble des valeurs possibles y compris les erreurs à l'exécution
  - $[\![\texttt{bool}]\!] = \{\texttt{true}, \texttt{false}, \bot_{runtime}\}$
  - $[\![\texttt{int}]\!] = \mathbb{Z} \cup \{\bot_{runtime}\}$
- Relations de comparaison des types :
  - Égalité
  - Compatibilité :
    - Généricité/Instanciation, Polymorphisme paramètrique
    - Sous-typage, Polymorphisme d'héritage
    - ...
- Règles de calcul (en miniML, Unification des types)

Mécanisme de typage

- Notion de type : Ensemble de valeurs pour lesquelles le programme a le même comportement
- Langage des types possibles : Syntaxe des types
  - Pour les expressions de miniML : `bool` et `int`
- Sémantique des types : Ensemble des valeurs possibles y compris les erreurs à l'exécution
  - $[\![\texttt{bool}]\!] = \{\texttt{true}, \texttt{false}, \bot_{runtime}\}$
  - $[\![\texttt{int}]\!] = \mathbb{Z} \cup \{\bot_{runtime}\}$
- Relations de comparaison des types :
  - Égalité
  - Compatibilité :
    - Généricité/Instanciation, Polymorphisme paramètrique
    - Sous-typage, Polymorphisme d'héritage
    - ...
- Règles de calcul (en miniML, Unification des types)

Mécanisme de typage

- Notion de type : Ensemble de valeurs pour lesquelles le programme a le même comportement
- Langage des types possibles : Syntaxe des types
  - Pour les expressions de miniML : `bool` et `int`
- Sémantique des types : Ensemble des valeurs possibles y compris les erreurs à l'exécution
  - $[\![\texttt{bool}]\!] = \{\texttt{true}, \texttt{false}, \bot_{runtime}\}$
  - $[\![\texttt{int}]\!] = \mathbb{Z} \cup \{\bot_{runtime}\}$
- Relations de comparaison des types :
  - Égalité
  - Compatibilité :
    - Généricité/Instanciation, Polymorphisme paramètrique
    - Sous-typage, Polymorphisme d'héritage
    - ...
- Règles de calcul (en miniML, Unification des types)

Mécanisme de typage

- Notion de type : Ensemble de valeurs pour lesquelles le programme a le même comportement
- Langage des types possibles : Syntaxe des types
  - Pour les expressions de miniML : `bool` et `int`
- Sémantique des types : Ensemble des valeurs possibles y compris les erreurs à l'exécution
  - $[\![\texttt{bool}]\!] = \{\texttt{true}, \texttt{false}, \bot_{runtime}\}$
  - $[\![\texttt{int}]\!] = \mathbb{Z} \cup \{\bot_{runtime}\}$
- Relations de comparaison des types :
  - Égalité
  - Compatibilité :
    - Généricité/Instanciation, Polymorphisme paramètrique
    - Sous-typage, Polymorphisme d'héritage
    - ...
- Règles de calcul (en miniML, Unification des types)

Analyseur statique : Principes généraux

- Programme qui détermine les propriétés d'un programme
- Langage hôte/support : Langage de programmation de l'analyseur
- Représenter le programme comme une donnée
- Représenter les propriétés comme des données
- Exprimer les règles de vérification comme des algorithmes
  - Résultats de la vérification (dont intermédiaires)
  - Pour chaque construction exécutable du langage, identifier :
    1. Variantes selon résultats intermédiaires
    2. Étapes dans chaque variante
    3. Contraintes entre les étapes

Analyseur statique : Principes généraux

- Programme qui détermine les propriétés d'un programme
- Langage hôte/support : Langage de programmation de l'analyseur
- Représenter le programme comme une donnée
- Représenter les propriétés comme des données
- Exprimer les règles de vérification comme des algorithmes
  - Résultats de la vérification (dont intermédiaires)
  - Pour chaque construction exécutable du langage, identifier :
    1. Variantes selon résultats intermédiaires
    2. Étapes dans chaque variante
    3. Contraintes entre les étapes

Analyseur statique : Principes généraux

- Programme qui détermine les propriétés d'un programme
- Langage hôte/support : Langage de programmation de l'analyseur
- Représenter le programme comme une donnée
- Représenter les propriétés comme des données
- Exprimer les règles de vérification comme des algorithmes
  - Résultats de la vérification (dont intermédiaires)
  - Pour chaque construction exécutable du langage, identifier :
    1. Variantes selon résultats intermédiaires
    2. Étapes dans chaque variante
    3. Contraintes entre les étapes

Application à miniML

- Arbre abstrait : voir vidéo séparée
- Syntaxe des types :

$$\begin{array}{rcl} \mathit{Type} & \rightarrow & \texttt{bool} \\ & | & \texttt{int} \end{array}$$

- Représentation des types et unification : voir vidéo séparée
- Algorithme de typage : voir vidéo séparée

Sémantique Axiomatique

- Objectif : Décrire formellement les mécanismes d'analyse des propriétés des programmes d'un langage
- Principe :
  - Exploiter la syntaxe du langage
  - Décrire les relations entre les constructions du langage et les propriétés
- Notation : Règles de déduction
- Jugement de typage : $\sigma \vdash e : \tau$
  - σ : environnement (association *Ident* / *Type*)
  - e : expression (*Expr*)
  - τ : type (*Type*)
- Partie haute : Étapes intermédiaires (appels récursifs dans typeur miniML)
- Partie basse : Construction traitée par la règle
- Principe de construction : Règles d'exécution congrue par la sémantique des types (façon classes d'équivalence)

miniML : Constantes et Accès identificateur

- Règles d'évaluation :

$$\frac{}{\gamma \vdash \mathit{entier} \Rightarrow \mathit{entier}} \qquad \frac{}{\gamma \vdash \mathit{booleen} \Rightarrow \mathit{booleen}}$$

- Règles de typage :

$$\frac{}{\sigma \vdash \mathit{entier} : \texttt{int}} \qquad \frac{}{\sigma \vdash \mathit{booleen} : \texttt{bool}}$$

- Identificateur : Accès à l'environnement
  - Transmission valeur associée :

$$\frac{x \in \gamma \quad \gamma(x) = v}{\gamma \vdash x \Rightarrow v}$$

  - Règle de typage associée :

$$\frac{x \in \sigma \quad \sigma(x) = \tau}{\sigma \vdash x : \tau}$$

miniML : Opérateur Unaire

- Étape préliminaire : Traitement du paramètre
- Variante 1 : Résultat correct du bon type

$$\frac{\gamma \vdash e \Rightarrow v \quad v \neq \bot \quad v \in dom\, op \quad v' = op\, v}{\gamma \vdash op\, e \Rightarrow v'}$$

- Variante 2 : Résultat erroné

$$\frac{\gamma \vdash e \Rightarrow v \quad v = \bot_c}{\gamma \vdash op\, e \Rightarrow \bot_c}$$

- Règle de typage associée :

$$\frac{\sigma \vdash e : \tau \quad \tau = dom\, op \quad \tau' = codom\, op}{\sigma \vdash op\, e : \tau'}$$

miniML : Opérateur Binaire Droite à Gauche

- Étapes préliminaires : Traitement des paramètres
- Variante 1 : Résultats corrects du bon type

$$\frac{\begin{array}{c}\gamma \vdash e_2 \Rightarrow v_2 \\ v_2 \neq \bot\end{array} \quad \begin{array}{c}\gamma \vdash e_1 \Rightarrow v_1 \\ v_1 \neq \bot\end{array} \quad v_1 \times v_2 \in dom\, op \quad v = v_1\ op\ v_2}{\gamma \vdash e_1\ op\ e_2 \Rightarrow v}$$

- Variante 2 : Résultat(s) erroné(s)

$$\frac{\gamma \vdash e_2 \Rightarrow v_2 \quad v_2 = \bot_c}{\gamma \vdash e_1\ op\ e_2 \Rightarrow \bot_c} \qquad \frac{\begin{array}{c}\gamma \vdash e_2 \Rightarrow v_2 \\ v_2 \neq \bot\end{array} \quad \gamma \vdash e_1 \Rightarrow v_1 \quad v_1 = \bot_c}{\gamma \vdash e_1\ op\ e_2 \Rightarrow \bot_c}$$

- Règle de typage associée :

$$\frac{\sigma \vdash e_1 : \tau_1 \quad \sigma \vdash e_2 : \tau_2 \quad \tau_1 \times \tau_2 = dom\, op \quad \tau = codom\, op}{\sigma \vdash e_1\ op\ e_2 : \tau}$$

Exemple de typage d'un programme miniML

- Prenons : $\sigma = \{v : \texttt{int}\}$
- Typons le programme miniML : $1 + v * 3$
- L'arbre est trop volumineux, décomposons en :

$$A = \frac{v \in \sigma \quad \sigma(v) = \texttt{int}}{\sigma \vdash v : \texttt{int}}$$

$$B = \frac{\begin{array}{cc} A & \texttt{int} \times \texttt{int} = dom* \\ \sigma \vdash 3 : \texttt{int} & \texttt{int} = codom* \end{array}}{\sigma \vdash v * 3 : \texttt{int}}$$

$$\frac{\begin{array}{cc} \sigma \vdash 1 : \texttt{int} & \texttt{int} \times \texttt{int} = dom+ \\ B & \texttt{int} = codom+ \end{array}}{\sigma \vdash 1 + v * 3 : \texttt{int}}$$

Correction du typage par rapport à l'exécution

-

Cours, TD, TP, mini-projet : miniC

- Types de données :
  - Types de base : `boolean`, `int`, `char`, `string`
  - Types structurées : n-uplets, Tableaux, Pointeurs, Enregistrements, Déclaration de types
- Algorithmes
  - Expressions sans effets de bord (sauf affectation)
  - Instructions : séquence, conditionnelle, répétition
  - Déclarations de variables avec et sans initialisation
  - Fonctions, Procédures avec récursivité

Projet : miniJava

- Classes et Interfaces génériques avec Instanciation explicite
- Constructeurs, Attributs et Méthodes d'Instances et de Classes avec droits d'accès et restriction d'héritage
- Polymorphisme d'héritage et Liaison tardive

Grammaires Attribuées : Principes généraux

- Objectif : Enricher la spécification de la syntaxe avec des éléments de sémantique
- Support : Règles de production
- Attributs sémantiques : Informations typées associées aux symboles (terminaux, non-terminaux)
- Équations sémantique : Relations entre les attributs des symboles d'une règle de production
- Question : Pour un programme donné, est il possible de calculer les valeurs des attributs sémantiques ?
- Solution : Calcul d'un point fixe sur les équations sémantiques
- Problème : Existence du point fixe en temps fini ? Raisonnable ?
- Approche : Restriction sur la forme des équations pour assurer la terminaison

Grammaires Attribuées : Principes généraux

- Objectif : Enricher la spécification de la syntaxe avec des éléments de sémantique
- Support : Règles de production
- Attributs sémantiques : Informations typées associées aux symboles (terminaux, non-terminaux)
- Équations sémantique : Relations entre les attributs des symboles d'une règle de production
- Question : Pour un programme donné, est il possible de calculer les valeurs des attributs sémantiques ?
- Solution : Calcul d'un point fixe sur les équations sémantiques
- Problème : Existence du point fixe en temps fini ? Raisonnable ?
- Approche : Restriction sur la forme des équations pour assurer la terminaison

Grammaires Attribuées : Principes généraux

- Objectif : Enricher la spécification de la syntaxe avec des éléments de sémantique
- Support : Règles de production
- Attributs sémantiques : Informations typées associées aux symboles (terminaux, non-terminaux)
- Équations sémantique : Relations entre les attributs des symboles d'une règle de production
- Question : Pour un programme donné, est il possible de calculer les valeurs des attributs sémantiques ?
- Solution : Calcul d'un point fixe sur les équations sémantiques
- Problème : Existence du point fixe en temps fini ? Raisonnable ?
- Approche : Restriction sur la forme des équations pour assurer la terminaison

Grammaires attribuées : Méthode

- Identifier les informations :
  - Disponibles avant l'analyse du programme : Contexte de l'analyse
  - Associées aux terminaux du programme : Informations lexicales
  - Associées à la structure de l'arbre de dérivation : Informations syntaxiques
  - Résultant de l'analyse sémantique
- Choisir des exemples représentatifs du langage
- Étiqueter :
  - Racine de l'arbre (axiome de la grammaire) : Informations de contexte
  - Feuilles de l'arbre (unités lexicales) : Informations lexicales
  - NœUds de l'arbre : Informations syntaxiques
- Étiqueter la racine avec les résultats attendus
- Identifier les relations entre les résultats attendus et les informations disponibles
- Introduire les attributs nécessaires pour les nœuds intermédiaires
- Définir et placer les actions sémantiques pour chaque nœud

Grammaires attribuées : Méthode

- Identifier les informations :
  - Disponibles avant l'analyse du programme : Contexte de l'analyse
  - Associées aux terminaux du programme : Informations lexicales
  - Associées à la structure de l'arbre de dérivation : Informations syntaxiques
  - Résultant de l'analyse sémantique
- Choisir des exemples représentatifs du langage
- Étiqueter :
  - Racine de l'arbre (axiome de la grammaire) : Informations de contexte
  - Feuilles de l'arbre (unités lexicales) : Informations lexicales
  - NœUds de l'arbre : Informations syntaxiques
- Étiqueter la racine avec les résultats attendus
- Identifier les relations entre les résultats attendus et les informations disponibles
- Introduire les attributs nécessaires pour les nœuds intermédiaires
- Définir et placer les actions sémantiques pour chaque nœud

Grammaires attribuées : Méthode

- Identifier les informations :
  - Disponibles avant l'analyse du programme : Contexte de l'analyse
  - Associées aux terminaux du programme : Informations lexicales
  - Associées à la structure de l'arbre de dérivation : Informations syntaxiques
  - Résultant de l'analyse sémantique
- Choisir des exemples représentatifs du langage
- Étiqueter :
  - Racine de l'arbre (axiome de la grammaire) : Informations de contexte
  - Feuilles de l'arbre (unités lexicales) : Informations lexicales
  - NœUds de l'arbre : Informations syntaxiques
- Étiqueter la racine avec les résultats attendus
- Identifier les relations entre les résultats attendus et les informations disponibles
- Introduire les attributs nécessaires pour les nœuds intermédiaires
- Définir et placer les actions sémantiques pour chaque nœud

Grammaires L-attribuées

- Objectif : Calcul pendant l'analyse syntaxique
- Hypothèse : Parcours de l'arbre de dérivation descendant puis ascendant de gauche à droite
- Remarque : Compatible avec analyse descendante récursive (grammaires $LL(k)$)
- Nature des attributs sémantiques des non terminaux :
  - Hérité (parcours descendant) : Calculé avant l'analyse du non terminal
  - Synthétisé (parcours ascendant) : Calculé pendant l'analyse du non terminal
- Forme des équations : Fonctions qui calculent la valeur des attributs
  - Synthétisés du symbole non terminal associé à la règle
  - Hérités des symboles non terminaux exploités par la règle
- Contrainte : Incompatible avec l'analyse ascendante (grammaires $LR(k)$)

Grammaires L-attribuées

- Objectif : Calcul pendant l'analyse syntaxique
- Hypothèse : Parcours de l'arbre de dérivation descendant puis ascendant de gauche à droite
- Remarque : Compatible avec analyse descendante récursive (grammaires $LL(k)$)
- Nature des attributs sémantiques des non terminaux :
  - Hérité (parcours descendant) : Calculé avant l'analyse du non terminal
  - Synthétisé (parcours ascendant) : Calculé pendant l'analyse du non terminal
- Forme des équations : Fonctions qui calculent la valeur des attributs
  - Synthétisés du symbole non terminal associé à la règle
  - Hérités des symboles non terminaux exploités par la règle
- Contrainte : Incompatible avec l'analyse ascendante (grammaires $LR(k)$)

Grammaires L-attribuées

- Objectif : Calcul pendant l'analyse syntaxique
- Hypothèse : Parcours de l'arbre de dérivation descendant puis ascendant de gauche à droite
- Remarque : Compatible avec analyse descendante récursive (grammaires $LL(k)$)
- Nature des attributs sémantiques des non terminaux :
  - Hérité (parcours descendant) : Calculé avant l'analyse du non terminal
  - Synthétisé (parcours ascendant) : Calculé pendant l'analyse du non terminal
- Forme des équations : Fonctions qui calculent la valeur des attributs
  - Synthétisés du symbole non terminal associé à la règle
  - Hérités des symboles non terminaux exploités par la règle
- Contrainte : Incompatible avec l'analyse ascendante (grammaires $LR(k)$)

Grammaires L-attribuées

- Objectif : Calcul pendant l'analyse syntaxique
- Hypothèse : Parcours de l'arbre de dérivation descendant puis ascendant de gauche à droite
- Remarque : Compatible avec analyse descendante récursive (grammaires $LL(k)$)
- Nature des attributs sémantiques des non terminaux :
  - Hérité (parcours descendant) : Calculé avant l'analyse du non terminal
  - Synthétisé (parcours ascendant) : Calculé pendant l'analyse du non terminal
- Forme des équations : Fonctions qui calculent la valeur des attributs
  - Synthétisés du symbole non terminal associé à la règle
  - Hérités des symboles non terminaux exploités par la règle
- Contrainte : Incompatible avec l'analyse ascendante (grammaires $LR(k)$)

Grammaires S-attribuées

- Objectif : Compatible avec analyseurs ascendants
- Uniquement des Attributs synthétisés
- Exécution des équations en fin de règle de production
- Exemples d'outils : `ocamlyacc`, `menhir`
- Qu'en est il des outils classiques de la famille `yacc` et `bison` ?
  - Utilisation de variables globales pour émuler les attributs hérités
  - Ajout de non-terminaux virtuels pour les actions sémantiques internes aux règles : Exécution de l'action sur la réduction de la règle
  - Introduit des conflits qui imposent la factorisation des règles : sous-ensemble des grammaires $LR(k)$
- Problèmes : Restrictions trop fortes pour la plupart des sémantiques
- Méthode associée :
  - Construction de l'arbre abstrait
  - Parcours de l'arbre abstrait pour les sémantiques plus complexes

Grammaires S-attribuées

- Objectif : Compatible avec analyseurs ascendants
- Uniquement des Attributs synthétisés
- Exécution des équations en fin de règle de production
- Exemples d'outils : `ocamlyacc`, `menhir`
- Qu'en est il des outils classiques de la famille `yacc` et `bison` ?
  - Utilisation de variables globales pour émuler les attributs hérités
  - Ajout de non-terminaux virtuels pour les actions sémantiques internes aux règles : Exécution de l'action sur la réduction de la règle
  - Introduit des conflits qui imposent la factorisation des règles : sous-ensemble des grammaires $LR(k)$
- Problèmes : Restrictions trop fortes pour la plupart des sémantiques
- Méthode associée :
  - Construction de l'arbre abstrait
  - Parcours de l'arbre abstrait pour les sémantiques plus complexes

Grammaires S-attribuées

- Objectif : Compatible avec analyseurs ascendants
- Uniquement des Attributs synthétisés
- Exécution des équations en fin de règle de production
- Exemples d'outils : `ocamlyacc`, `menhir`
- Qu'en est il des outils classiques de la famille `yacc` et `bison` ?
  - Utilisation de variables globales pour émuler les attributs hérités
  - Ajout de non-terminaux virtuels pour les actions sémantiques internes aux règles : Exécution de l'action sur la réduction de la règle
  - Introduit des conflits qui imposent la factorisation des règles : sous-ensemble des grammaires $LR(k)$
- Problèmes : Restrictions trop fortes pour la plupart des sémantiques
- Méthode associée :
  - Construction de l'arbre abstrait
  - Parcours de l'arbre abstrait pour les sémantiques plus complexes

Exemple : Typage de miniML

- Attribut hérité : Environnement de typage
- Attribut synthétisé : Type de l'expression
- Action sémantique : Régle de typage associée à la régle de production

$$
\begin{array}{lcl}
E & \rightarrow & \#1\ E + \#2\ T\ \#3 \\
\#1 & : & E_1.env = E.env \\
\#2 & : & T.env = E.env \\
\#3 & : & E.type = Typage(+, E_1.type, T.type)
\end{array}
$$

$$
\begin{array}{lcl}
E & \rightarrow & \#1\ T\ \#2 \\
\#1 & : & T.env = E.env \\
\#2 & : & E.type = T.type
\end{array}
$$

$$
\begin{array}{lcl}
T & \rightarrow & \#1\ T * \#2\ F\ \#3 \\
\#1 & : & T_1.env = T.env \\
\#2 & : & F.env = T.env \\
\#3 & : & E.type = Typage(*, T_1.type, F.type)
\end{array}
$$

$$
\begin{array}{lcl}
T & \rightarrow & \#1\ F\ \#2 \\
\#1 & : & F.env = T.env \\
\#2 & : & T.type = F.type
\end{array}
$$

$$
\begin{array}{lcl}
F & \rightarrow & \#1\ (\ E\ )\ \#2 \\
\#1 & : & E.env = F.env \\
\#2 & : & F.type = E.type
\end{array}
$$

$$
\begin{array}{lcl}
F & \rightarrow & \#1\ -F\ \#2 \\
\#1 & : & F_1.env = F.env \\
\#2 & : & F.type = Typage(-, F_1.type)
\end{array}
$$

$$
\begin{array}{lcl}
F & \rightarrow & \texttt{entier}\ \#1 \\
\#1 & : & F.type = \texttt{int}
\end{array}
$$

$$
\begin{array}{lcl}
F & \rightarrow & \texttt{ident}\ \#1 \\
\#1 & : & F.type = Recherche(F.env, ident.texte)
\end{array}
$$

Exemple : Typage de miniML

- Élimination de la récursivité à gauche
- Attributs hérités supplémentaires : `typeh`

$$
\begin{array}{lcl}
E & \to & \#1\ T\ \#2\ ST\ \#3 \\
\#1 & : & T.env = E.env \\
\#2 & : & ST.env = E.env \\
 & & ST.typeh = T.type \\
\#3 & : & E.type = ST.type
\end{array}
$$

$$
\begin{array}{lcl}
ST & \to & \#1\ +\ T\ \#2\ ST\ \#3 \\
\#1 & : & ST_1.env = ST.env \\
\#2 & : & T.env = ST.env \\
 & & ST_1.typeh = Typage(+, ST.typeh, T.type) \\
\#3 & : & ST.type = ST_1.type
\end{array}
$$

$$
\begin{array}{lcl}
ST & \to & \#1 \\
\#1 & : & ST.type = ST.typeh
\end{array}
$$

$$
\begin{array}{lcl}
T & \to & \#1\ F\ \#2\ SF\ \#3 \\
\#1 & : & F.env = T.env \\
\#2 & : & SF.env = T.env \\
 & & SF.typeh = F.type \\
\#3 & : & T.type = SF.type
\end{array}
$$

$$
\begin{array}{lcl}
SF & \to & \#1\ *\ F\ \#2\ SF\ \#3 \\
\#1 & : & SF_1.env = SF.env \\
\#2 & : & F.env = SF.env \\
 & & SF_1.typeh = Typage(*, SF.typeh, F.type) \\
\#3 & : & SF.type = SF_1.type
\end{array}
$$

$$
\begin{array}{lcl}
SF & \to & \#1 \\
\#1 & : & SF.type = SF.typeh
\end{array}
$$

Exemple : Exécution de miniML

- Attribut hérité : Environnement d'exécution
- Attribut synthétisé : Valeur de l'expression
- Action sémantique : Régle d'exécution associée à la régle de production

Rappel : Arbres de dérivation (syntaxique)

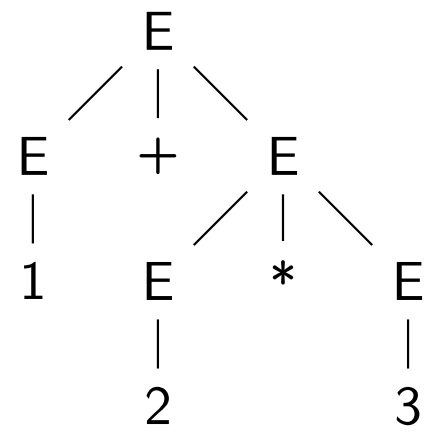
- Objectif : Représenter la structure du mot induite par les règles de production lors d'une dérivation
- Feuilles de l'arbre : Terminaux composant le mot
- Racine de l'arbre : Axiome
- Nœuds de l'arbre : Non-terminaux apparaissant dans la dérivation
- Branches de l'arbre : Règles de production

Exemple :
Arbre de dérivation pour 1 + 2 * 3

Rappel : Arbres de dérivation (syntaxique)

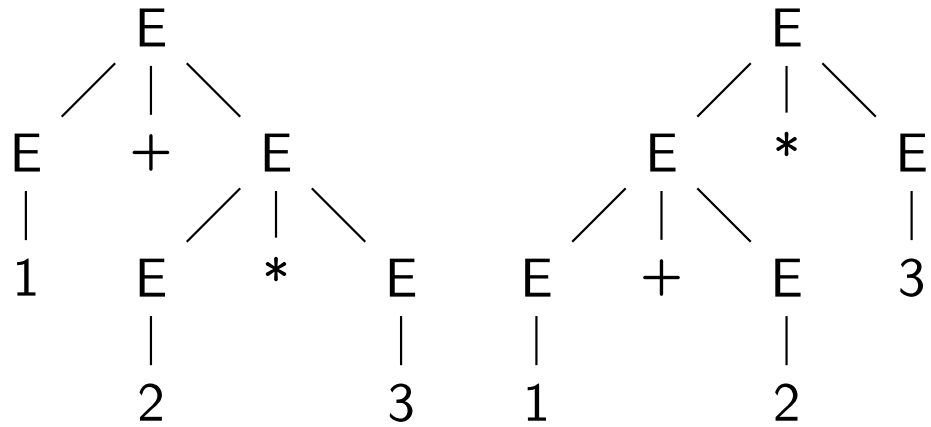
- Objectif : Représenter la structure du mot induite par les règles de production lors d'une dérivation
- Feuilles de l'arbre : Terminaux composant le mot
- Racine de l'arbre : Axiome
- Nœuds de l'arbre : Non-terminaux apparaissant dans la dérivation
- Branches de l'arbre : Règles de production

Exemple :
Arbre de dérivation pour `1 + 2 * 3`



Rappel : Grammaire et langage ambigu

- Une grammaire est ambigue s'il existe plusieurs arbres de dérivation distincts pour un même mot
- Exemple : Arbres de dérivation pour `1 + 2 * 3`



- Un langage est ambigu si toutes les grammaires le représentant sont ambigues

Grammaires pour les expressions

- Associativité codée par récursivité
- Priorité codée par imbrication des régles
- Grammaire $LR(k)$:

$$
\begin{array}{llll}
E & \rightarrow & E + T & F \rightarrow (\ E\ ) \\
E & \rightarrow & T & F \rightarrow -F \\
T & \rightarrow & T * F & F \rightarrow \texttt{entier} \\
T & \rightarrow & F & F \rightarrow \texttt{ident}
\end{array}
$$

Arbre associé :

- Grammaire $LL(k)$:

$$
\begin{array}{lll}
E & \rightarrow & T\ ST \\
ST & \rightarrow & +\ T\ ST \\
ST & \rightarrow & \Lambda \\
T & \rightarrow & F\ SF \\
SF & \rightarrow & *\ F\ SF \\
SF & \rightarrow & \Lambda
\end{array}
$$

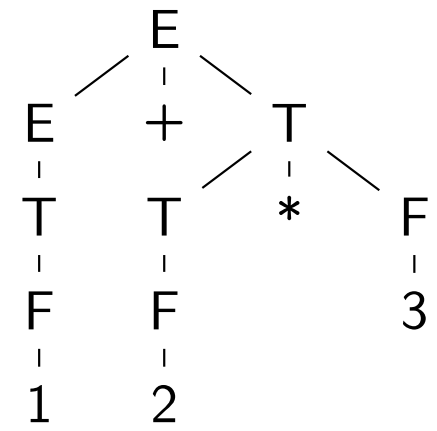
Arbre associé :

Grammaires pour les expressions

- Associativité codée par récursivité
- Priorité codée par imbrication des régles
- Grammaire $LR(k)$:

$$
\begin{array}{lcllcl}
E & \rightarrow & E + T & F & \rightarrow & (E) \\
E & \rightarrow & T & F & \rightarrow & - F \\
T & \rightarrow & T * F & F & \rightarrow & \texttt{entier} \\
T & \rightarrow & F & F & \rightarrow & \texttt{ident}
\end{array}
$$

Arbre associé :



- Grammaire $LL(k)$:

$$
\begin{array}{lcl}
E & \rightarrow & T\ ST \\
ST & \rightarrow & +\ T\ ST \\
ST & \rightarrow & \Lambda \\
T & \rightarrow & F\ SF \\
SF & \rightarrow & *\ F\ SF \\
SF & \rightarrow & \Lambda
\end{array}
$$

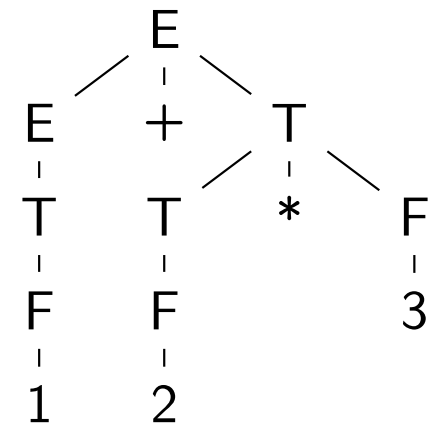
Arbre associé :

Grammaires pour les expressions

- Associativité codée par récursivité
- Priorité codée par imbrication des régles
- Grammaire $LR(k)$:

$$
\begin{array}{lcllcl}
E & \to & E + T & F & \to & (E) \\
E & \to & T & F & \to & - F \\
T & \to & T * F & F & \to & \texttt{entier} \\
T & \to & F & F & \to & \texttt{ident}
\end{array}
$$

Arbre associé :



- Grammaire $LL(k)$:

$$
\begin{array}{lcl}
E & \to & T\ ST \\
ST & \to & + T\ ST \\
ST & \to & \Lambda \\
T & \to & F\ SF \\
SF & \to & * F\ SF \\
SF & \to & \Lambda
\end{array}
$$

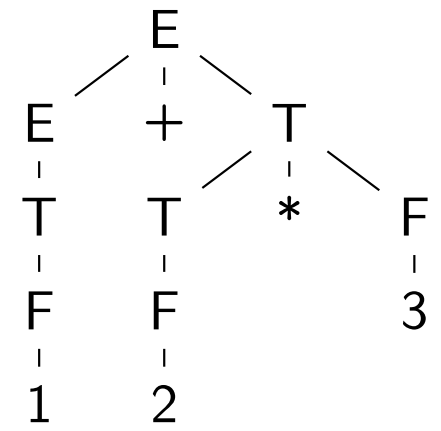
Arbre associé :

Grammaires pour les expressions

- Associativité codée par récursivité
- Priorité codée par imbrication des régles
- Grammaire $LR(k)$:

$$
\begin{array}{lcllcl}
E & \to & E + T & F & \to & (E) \\
E & \to & T & F & \to & - F \\
T & \to & T * F & F & \to & \texttt{entier} \\
T & \to & F & F & \to & \texttt{ident}
\end{array}
$$

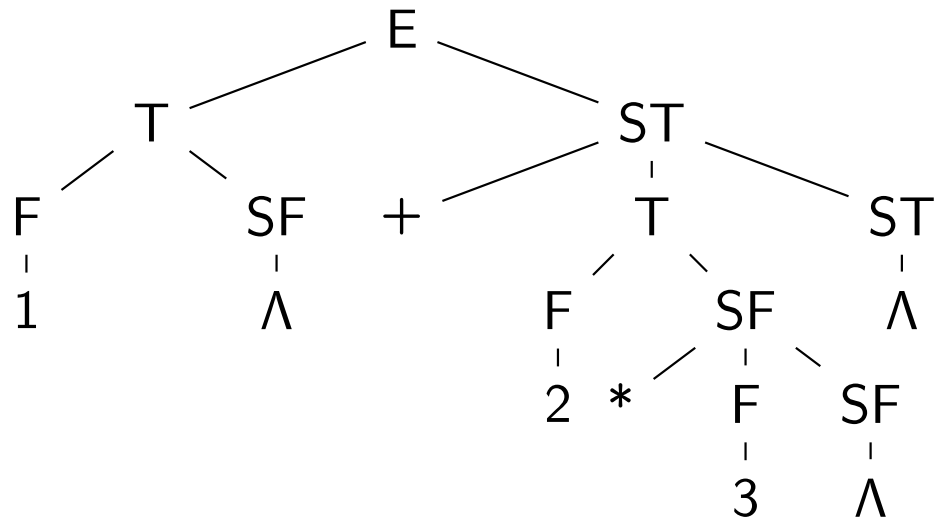
Arbre associé :



- Grammaire $LL(k)$:

$$
\begin{array}{lcl}
E & \to & T\ ST \\
ST & \to & +\ T\ ST \\
ST & \to & \Lambda \\
T & \to & F\ SF \\
SF & \to & *\ F\ SF \\
SF & \to & \Lambda
\end{array}
$$

Arbre associé :



Construction de l'Arbre Abstrait

- Arbre de dérivation / Arbre syntaxique : Construit automatiquement à partir de la structure de la grammaire
  - Satisfaisant pour les grammaires $LR(k)$
  - Déformé par l'élimination de la récursivité à gauche et la factorisation pour les grammaires $LL(k)$
- Arbre abstrait :
  - Support pour les étapes suivantes d'analyse sémantique
  - Simplification de l'arbre syntaxique (élimination des nœuds inutiles)
  - Réparation des déformations $LL(k)$
- Modèle de donnée pour l'analyse sémantique : méta-modèle
- Exemple des expressions : voir TD GLS Patron Visiteur

Construction de l'Arbre Abstrait

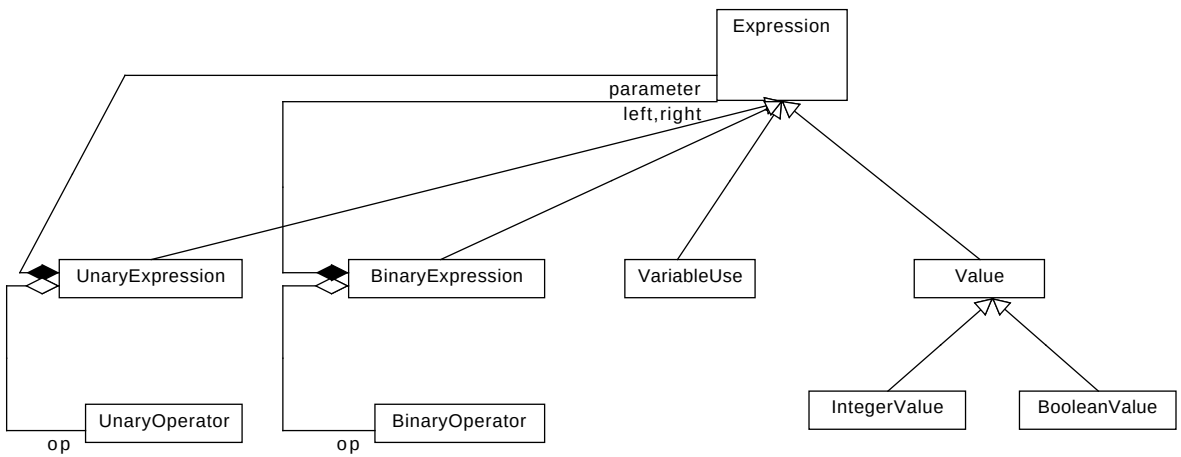
- Arbre de dérivation / Arbre syntaxique : Construit automatiquement à partir de la structure de la grammaire
  - Satisfaisant pour les grammaires $LR(k)$
  - Déformé par l'élimination de la récursivité à gauche et la factorisation pour les grammaires $LL(k)$
- Arbre abstrait :
  - Support pour les étapes suivantes d'analyse sémantique
  - Simplification de l'arbre syntaxique (élimination des nœuds inutiles)
  - Réparation des déformations $LL(k)$
- Modèle de donnée pour l'analyse sémantique : méta-modèle
- Exemple des expressions : voir TD GLS Patron Visiteur

Construction de l'Arbre Abstrait

- Arbre de dérivation / Arbre syntaxique : Construit automatiquement à partir de la structure de la grammaire
  - Satisfaisant pour les grammaires $LR(k)$
  - Déformé par l'élimination de la récursivité à gauche et la factorisation pour les grammaires $LL(k)$
- Arbre abstrait :
  - Support pour les étapes suivantes d'analyse sémantique
  - Simplification de l'arbre syntaxique (élimination des nœuds inutiles)
  - Réparation des déformations $LL(k)$
- Modèle de donnée pour l'analyse sémantique : méta-modèle
- Exemple des expressions : voir TD GLS Patron Visiteur

Construction de l'Arbre Abstrait

- Arbre de dérivation / Arbre syntaxique : Construit automatiquement à partir de la structure de la grammaire
  - Satisfaisant pour les grammaires $LR(k)$
  - Déformé par l'élimination de la récursivité à gauche et la factorisation pour les grammaires $LL(k)$
- Arbre abstrait :
  - Support pour les étapes suivantes d'analyse sémantique
  - Simplification de l'arbre syntaxique (élimination des nœuds inutiles)
  - Réparation des déformations $LL(k)$
- Modèle de donnée pour l'analyse sémantique : méta-modèle
- Exemple des expressions : voir TD GLS Patron Visiteur



Gestion de la Table des Symboles

- Objectif :
  1. Lier les définitions et les utilisations des identificateurs
  2. Collecter toutes les informations associées aux identificateurs :
     - contenues initialement dans le programme
     - calculées par la sémantique
- Exemple : Environnement d'exécution et de typage de miniML
- Deux approches possibles :
  1. Manipulation explicite d'un dictionnaire (environnement de miniML)
  2. Construction de liens dans l'arbre abstrait
- Gestion de la portée des définitions : Table des symboles hiérarchique
  - Une table pour chaque espace de noms
  - Relations entre tables qui correspondent à l'inclusion (les recouvrements de portée) des espaces de noms

Architecture de la table des symboles

- Modèle de données fourni :