

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 Dirigés 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

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
 - ▶ Reconnaître 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

[illegible]

Analyse lexicale

- Informations élémentaires : **commentaire**, **nombre**, **identificateur**, `.` (unités lexicales)

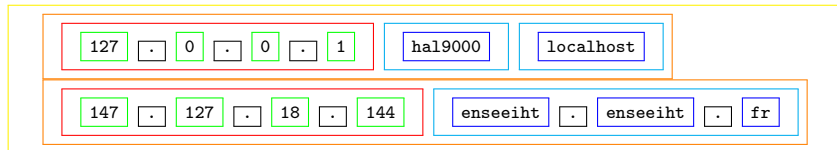
- Résultat de l'analyse lexicale :

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

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

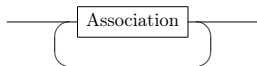
Analyse syntaxique

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

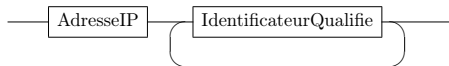


- ▶ Spécification des unités syntaxiques : Grammaires (notation de Conway)

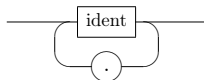
Document



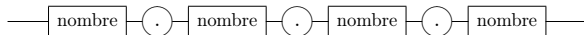
Association



IdentificateurQualifie

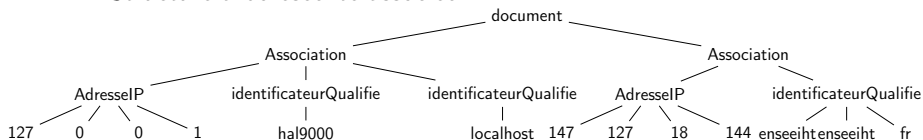


AdresseIP



Analyse sémantique

- Structure arborescente associée :



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

hal9000

localhost

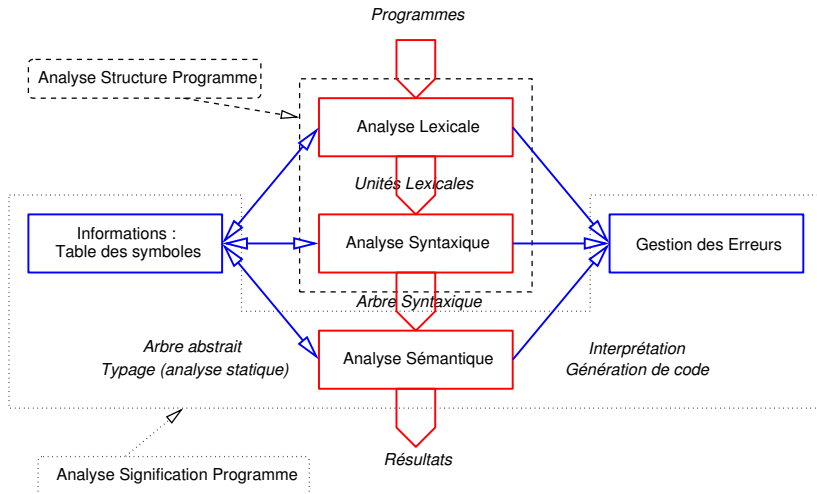
enseiht . enseiht . fr

127 . 0 . 0 . 1

127 . 0 . 0 . 1

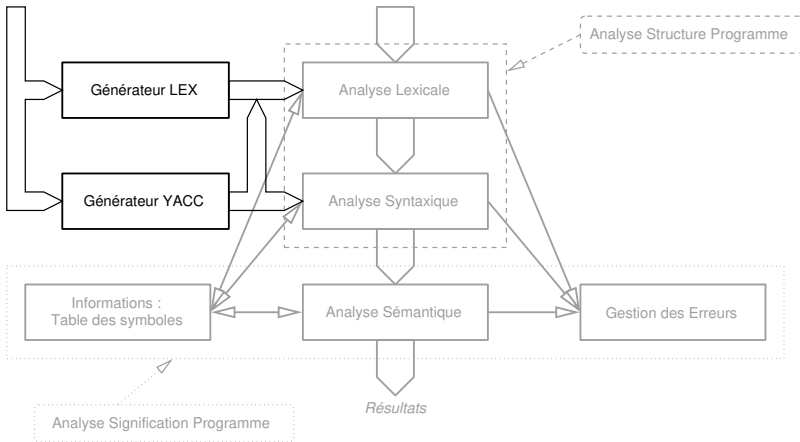
147 . 127 . 18 . 144

Structure d'un outil



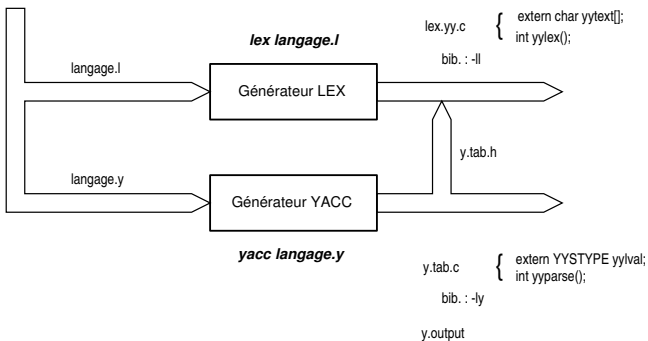
Exemple lex et yacc

Description Langage



Exemple lex et yacc

Description Langage



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 : Élément 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

- ▶ 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

► Expressions :

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

miniML

► Partie Fonctionnelle :

$$\begin{array}{lcl} Expr & \rightarrow & \dots \\ & | & \text{let } Ident = Expr \text{ in } Expr \\ & | & \text{if } Expr \text{ then } Expr \text{ else } Expr \\ & | & \text{fun } Ident \rightarrow Expr \\ & | & (Expr) Expr \\ & | & \text{let rec } Ident = Expr \text{ in } Expr \end{array}$$

► Partie Impératives :

$$\begin{array}{lcl} Expr & \rightarrow & \text{ref } Expr \\ & | & ! Expr \\ & | & Expr := Expr \\ & | & Expr ; Expr \\ & | & \text{while } Expr \text{ do } Expr \text{ done} \end{array}$$

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

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 :

$$\begin{array}{ccc} \textit{Valeur} & \rightarrow & \textit{Const} \\ | & & \perp \end{array}$$

- ▶ 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, \dots, J_n et J des jugements :

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

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

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

- 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

$$\frac{x \notin \gamma}{\gamma \vdash x \Rightarrow \perp_{\textit{undef}}}$$

miniML : Opérateur Unaire

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

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

- ▶ Variante 2 : Résultat erroné

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

- ▶ Variante 3 : Résultat correct du mauvais type

$$\frac{\gamma \vdash e \Rightarrow v \quad v \neq \perp \quad v \notin \text{dom } op}{\gamma \vdash op \, e \Rightarrow \perp_{type}}$$

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

$$\frac{\begin{array}{l} \gamma \vdash e_1 \Rightarrow v_1 \quad v_1 \neq \perp \\ \gamma \vdash e_2 \Rightarrow v_2 \quad v_2 \neq \perp \end{array} \quad v_1 \times v_2 \in \text{dom } op \quad v = v_1 \text{ op } v_2}{\gamma \vdash e_1 \text{ op } e_2 \Rightarrow v}$$

- ▶ Variante 2 : Résultat(s) erroné(s)

$$\frac{\gamma \vdash e_1 \Rightarrow v_1 \quad v_1 = \perp_c}{\gamma \vdash e_1 \text{ op } e_2 \Rightarrow \perp_c} \quad \frac{\gamma \vdash e_2 \Rightarrow v_2 \quad v_2 = \perp_c}{\gamma \vdash e_1 \text{ op } e_2 \Rightarrow \perp_c}$$

- ▶ 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

$$\frac{\gamma \vdash e_1 \Rightarrow v_1 \quad \gamma \vdash e_2 \Rightarrow v_2 \quad v_1 \neq \perp \quad v_2 \neq \perp \quad v_1 \times v_2 \notin \text{dom } op}{\gamma \vdash e_1 \text{ op } e_2 \Rightarrow \perp_{\text{type}}}$$

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

$$\frac{\gamma \vdash e_2 \Rightarrow v_2 \quad \gamma \vdash e_1 \Rightarrow v_1 \quad v_1 \times v_2 \in \text{dom } op \quad v = v_1 \text{ op } v_2}{\gamma \vdash e_1 \text{ op } e_2 \Rightarrow v}$$

Attention : Cette règle n'impose pas d'ordre

- Variante 2 : Résultat(s) erroné(s)

$$\frac{\gamma \vdash e_2 \Rightarrow v_2 \quad v_2 = \perp_c}{\gamma \vdash e_1 \text{ op } e_2 \Rightarrow \perp_c} \quad \frac{\gamma \vdash e_2 \Rightarrow v_2 \quad \gamma \vdash e_1 \Rightarrow v_1 \quad v_1 = \perp_c \quad v_2 \neq \perp}{\gamma \vdash e_1 \text{ op } e_2 \Rightarrow \perp_c}$$

- 2 erreurs ne peuvent plus se produire en même temps
- Variante 3 : Résultat correct du mauvais type

$$\frac{\gamma \vdash e_1 \Rightarrow v_1 \quad \gamma \vdash e_2 \Rightarrow v_2 \quad v_1 \neq \perp \quad v_2 \neq \perp \quad v_1 \times v_2 \notin \text{dom } op}{\gamma \vdash e_1 \text{ op } e_2 \Rightarrow \perp_{\text{type}}}$$

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 :

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

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

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

Définitions récursives

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

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

- ▶ Rendons f visible dans e_1 :

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

- ▶ Question : Est ce bien fondé ?

- ▶ Remarque :

$\text{let rec } f = e_1 \text{ in } e_2 \equiv \text{let } f = \text{let rec } f = e_1 \text{ in } e_1 \text{ in } e_2$

- ▶ Exploitions cette relation :

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

- ▶ Est ce un progrès ?

Définitions récursives

- Si nous le faisons une seconde fois :

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

- Si e_1 s'évalue en une fonction $\langle \text{fun } x \rightarrow e_3, \gamma_{\text{def}} \rangle$
- 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

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

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

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

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

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, ...)

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
 - ▶ $\llbracket \text{bool} \rrbracket = \{\text{true}, \text{false}, \perp_{\text{runtime}}\}$
 - ▶ $\llbracket \text{int} \rrbracket = \mathbb{Z} \cup \{\perp_{\text{runtime}}\}$
- ▶ Relations de comparaison des types :
 - ▶ Égalité
 - ▶ Compatibilité :
 - ▶ Généricité/Instanciation, Polymorphisme paramétrique
 - ▶ 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

Application à miniML

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

$$\begin{array}{ccc} Type & \rightarrow & \text{bool} \\ & | & \text{int} \end{array}$$

- ▶ 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 :

$$\frac{}{\gamma \vdash \text{entier} \Rightarrow \text{entier}} \quad \frac{}{\gamma \vdash \text{booleen} \Rightarrow \text{booleen}}$$

- ▶ Règles de typage :

$$\frac{}{\sigma \vdash \text{entier} : \text{int}} \quad \frac{}{\sigma \vdash \text{booleen} : \text{bool}}$$

- ▶ 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

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

- ▶ Variante 2 : Résultat erroné

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

- ▶ Règle de typage associée :

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

miniML : Opérateur Binaire Droite à Gauche

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

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

- ▶ Variante 2 : Résultat(s) erroné(s)

$$\frac{\gamma \vdash e_2 \Rightarrow v_2 \quad v_2 = \perp_c}{\gamma \vdash e_1 \text{ op } e_2 \Rightarrow \perp_c} \quad \frac{\gamma \vdash e_2 \Rightarrow v_2 \quad v_2 \neq \perp \quad \gamma \vdash e_1 \Rightarrow v_1 \quad v_1 = \perp_c}{\gamma \vdash e_1 \text{ op } e_2 \Rightarrow \perp_c}$$

- ▶ Règle de typage associée :

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

Exemple de typage d'un programme miniML

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

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

$$B = \frac{\begin{array}{c} A \qquad \text{int} \times \text{int} = \text{dom} * \\ \sigma \vdash 3 : \text{int} \qquad \text{int} = \text{codom} * \end{array}}{\sigma \vdash v * 3 : \text{int}}$$

$$\frac{\begin{array}{c} \sigma \vdash 1 : \text{int} \qquad \text{int} \times \text{int} = \text{dom} + \\ B \qquad \text{int} = \text{codom} + \end{array}}{\sigma \vdash 1 + v * 3 : \text{int}}$$







Grammaires Attribuées : Principes généraux

- ▶ Objectif : Enrichir 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 q'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 ? Raisnable ?
- ▶ Approche : Restriction sur la forme des équations pour assurer la terminaison

Grammaire L-attribuées

- ▶ Objectif : Calcul pendant l'analyse syntaxique
- ▶ Contrainte : Analyseur Descendant
- ▶ Attributs synthétisés : Attributs d'un non terminal dont la valeur est calculée pendant l'analyse de ce non terminal
- ▶ Attributs hérités : Attributs d'un non terminal dont la valeur est calculée avant l'analyse de ce non terminal
- ▶ Forme des équations : Exploitation de fonctions qui calculent la valeur des attributs
 - ▶ Synthétisés du symbole non terminal décrit par la règle
 - ▶ Hérités des symboles non terminaux exploités par la règle
- ▶ Contraintes :
 - ▶ Calcul de gauche à droite :
 - ▶ Calcul de haut en bas
 - ▶ Calcul de bas en haut



Exemple : Typage de miniML



Exemple : Exécution de miniM







