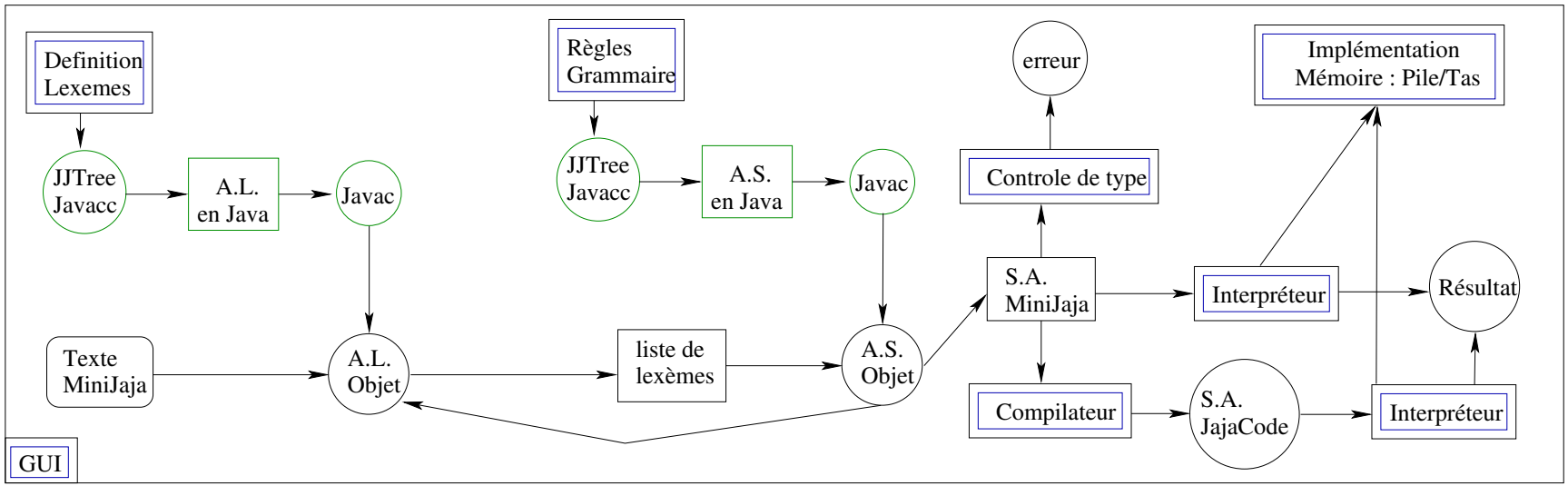
Compilation et Interprétations



F. Bouquet

Master S&T - Mention Informatique

2015 - 2016

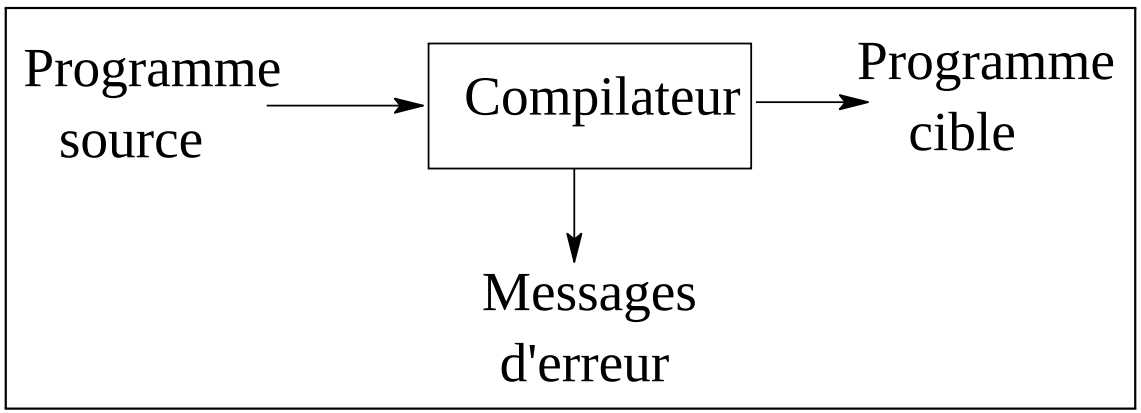
Université de Franche-Comté

Plan

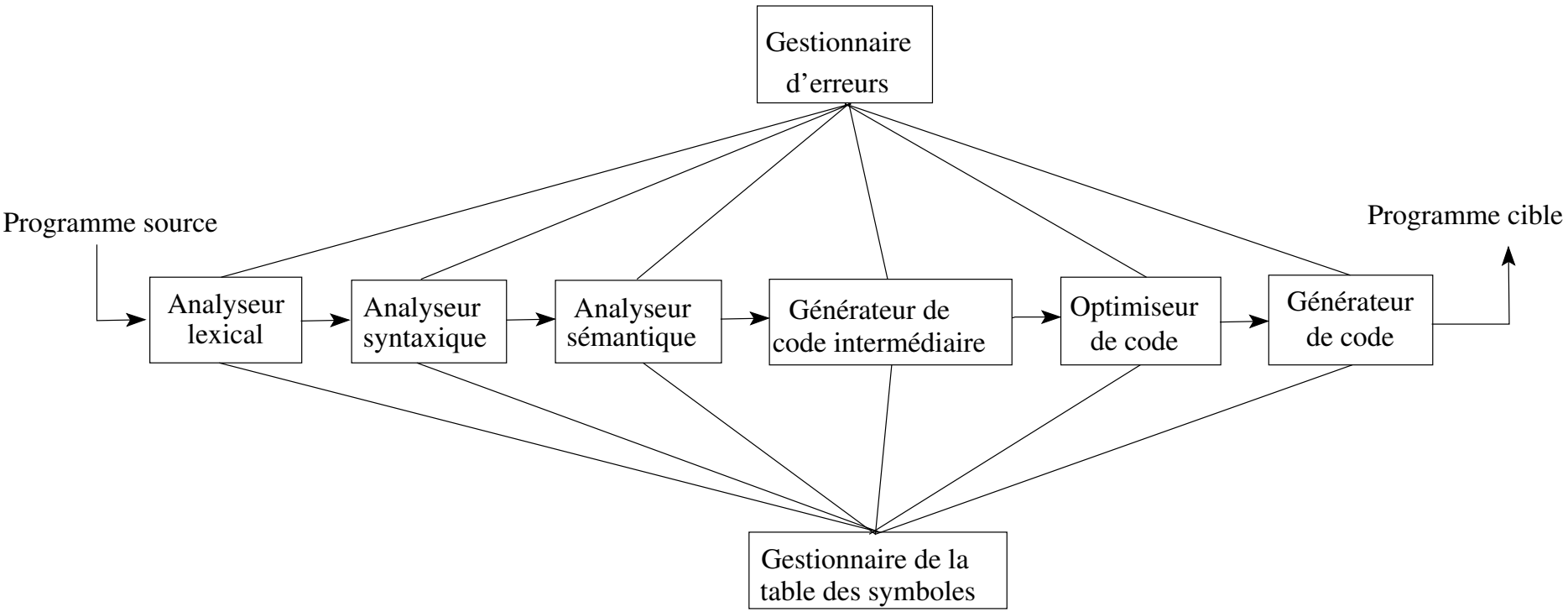
Compilateur

Traducteur (Source / Destination) :

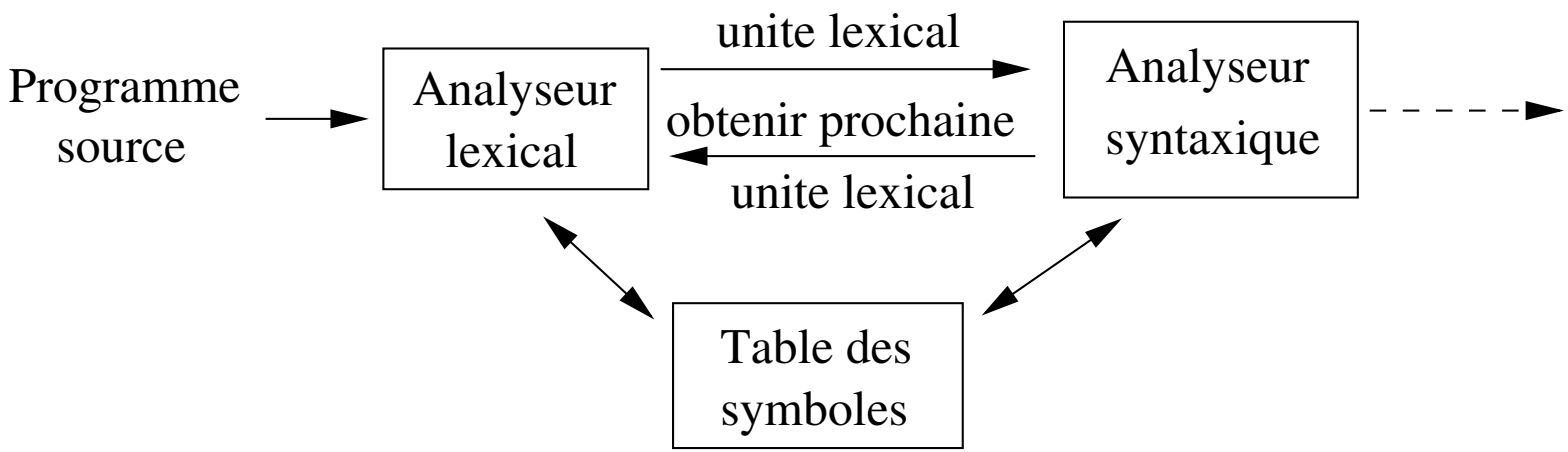
- Langage de hauts niveaux $\rightarrow$ Assembleur
- Système de composition de textes
- Compilateur de silicium
- Interprètes de requêtes



Phase de la compilation



Analyseur lexical



Analyseur lexical

- **Rôle :**
  - Lire le texte source
  - Relier les messages d'erreur du compilateur à la source
- **Tâches annexes :**
  - Éliminations des commentaires et espaces
  - Traitement des macros
- **Pourquoi :**
  - Conception simplifiée (2 Analyseurs : Syntaxique / Lexicale)
  - Spécialisation de la lecture (Tampons E/S, Traitements)
  - Portabilité (particularités de l'alphabet et anomalies)

Analyseur Lexical

- **But :**
  A partir des **modèles** définies par la grammaire il donne les **unités lexicales** qui vont servir à l'analyse syntaxique lorsqu'il identifie dans le texte le **lexème** correspondant.
- **Construction des lexèmes :**
  - Expressions régulières
  - Automate d'états finis
- **Erreur :**
  - Rare au niveau lexical
  - Propager celle de la syntaxe

Spécification des unités lexicales

- Décomposition d'une chaîne s :
  - **Préfixe** de s, **Suffixe** de s, **Sous-chaîne** de s
  - Préfixe, suffixe, sous-chaîne propre de s
  - **Sous-suite** de s
- Opérateurs :
  - **Union** de L et M, notée $L \cup M$
  - **Concaténation** de L et M, notée LM
  - **Fermeture** de **Kleene** de L, notée L^*
  - Fermeture **positive** de L, notée L^+
- Expressions régulières

Grammaire

- **Pourquoi :**
  - Spécification syntaxique précise, mais compréhensible
  - Structuration du langage
  - Évolution plus facile
  - Classes de grammaires avec de bonnes propriétés

- **Type de grammaire :**
  Grammaire non contextuelle, notation **B**ackus-**N**aur **F**orm (**BNF**).

Analyseur syntaxique

- **Rôle :**
  - Vérification de la grammaire
  - Construction des arbres d'analyse
- **Type d'analyses :**
  - Descendante (Racine $\rightarrow$ Feuilles)
  - Ascendante (Feuilles $\rightarrow$ Racine)
- **Sous-classes de grammaire,** $G =< X, V, P, S >$ **:**
  - LL(k) : **L**eft most derivation pour la construction de l'arbre
  - LR(k) : **R**ight most derivation pour la construction de l'arbre
    - **L**eft to right pour l'analyse du texte
    - **k** symboles de la chaîne d'entrée

Analyse descendante LL(1)

Propriétés :

- Non récursive à gauche
- Non ambigüe

Reconnaissance :

- Automate à pile
- Table d'analyse syntaxique :
  - Chaque colonne correspond à un **symbole terminal** $\in$ **X**
  - Chaque ligne correspond à un **symbole non terminal** $\in$ **V**
  - Une case correspond :
    - à une **règle de production** $\in$ **P**
    - à un cas d'erreur

Analyse Ascendante (SLR, LR, LALR...)

- **Rôle :**
  - Construire l'arbre à partir des feuilles
  - Utilisation d'une pile
  - On utilise les règles à l'envers (Droite $\rightarrow$ Gauche)
  - En finale, on obtient l'axiome S.
- **A chaque étape, on peut :**
  - Faire un **décalage** traiter un lexème
  - Faire une **Réduction** utiliser une règle

Traitement des erreurs

- **Deux situations permettent de détecter des erreurs :**
  - Le sommet de la pile $\neq$ du symbole terminal courant
  - On arrive sur $M[Y, a]$ vide (ou contenant *Erreur*)
- **Stratégies de poursuite de l'analyse :**
  - *Récupération en mode panique*, on cherche un point d'entrée à nouveau correct :
    - Suppression du sommet de pile et poursuite
    - Lecture source jusqu'à trouver une unité lexicale compatible avec le sommet de la pile
  - *Récupération au niveau du syntagme*, on utilise les cases erronées de la table pour indiquer les routines de traitement.

Table des symboles

But : Garder la trace de la portée et la liaison des noms

Une **entrée** doit être composée :

- Nom
- Attributs

Action sur les éléments :

- Rechercher,
- Ajouter,
- Supprimer

Structure de données en liste

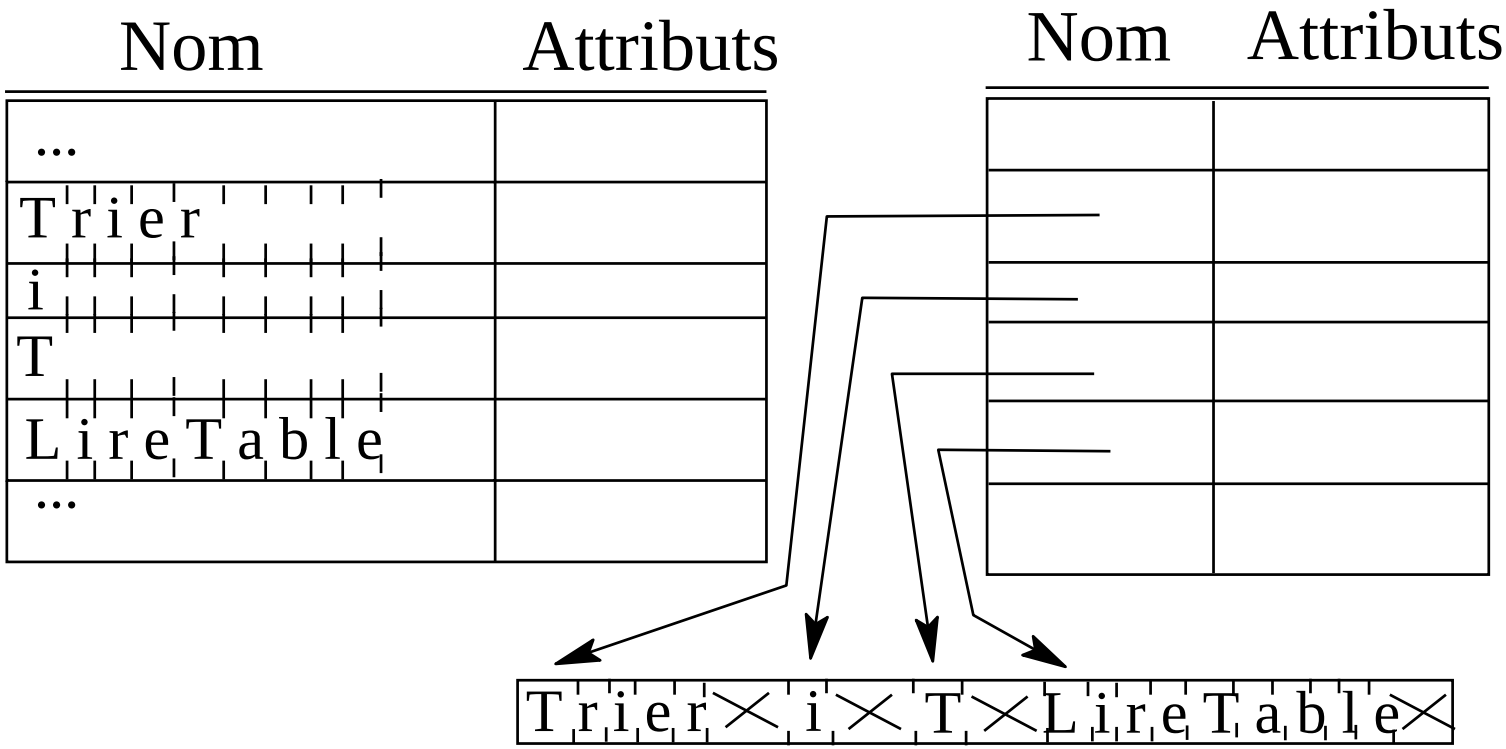
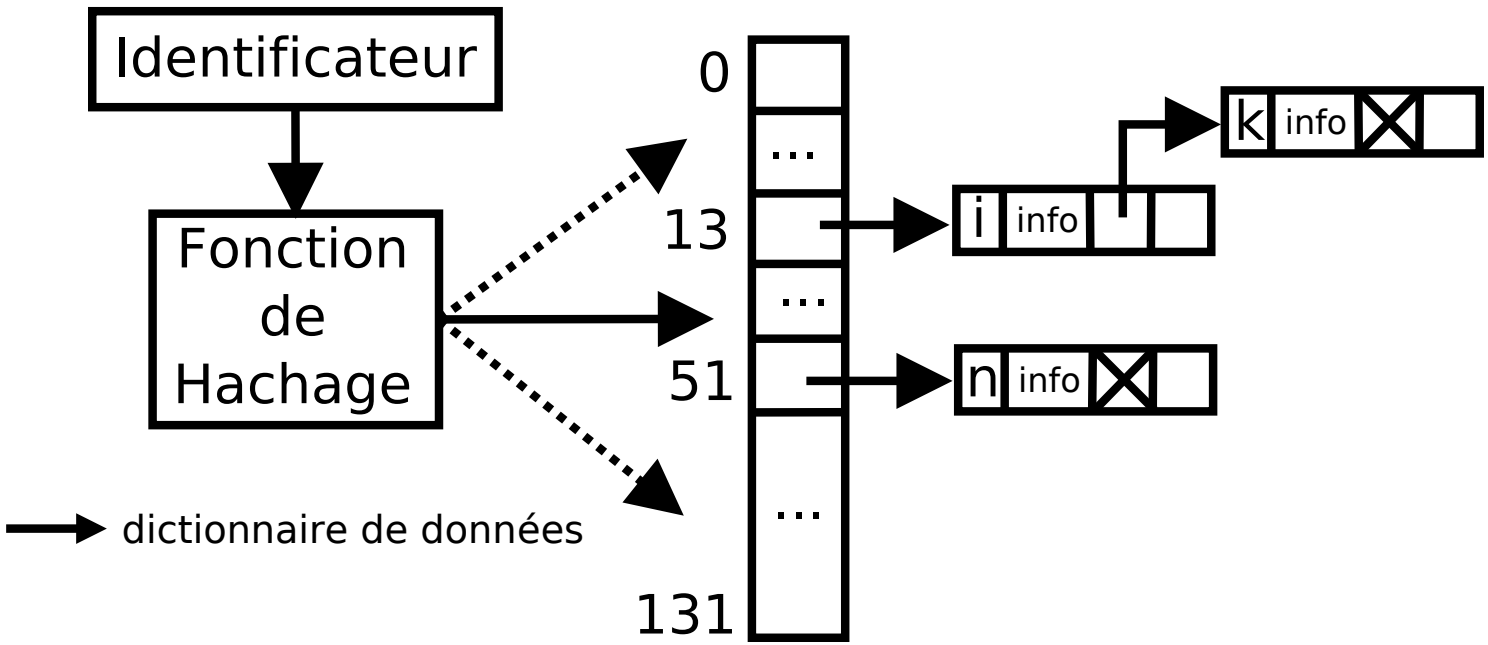


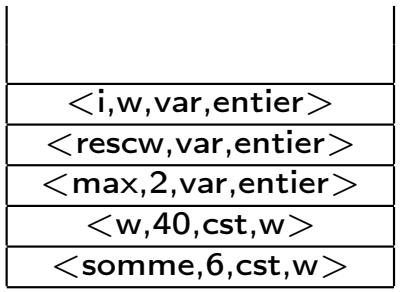
Table de Hachage



Pile d'exécution

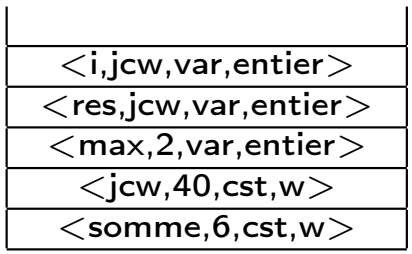
Rôle :

- Mémoire pour les variables
- Sauvegarde de l'environnement
- Calcul temporaire

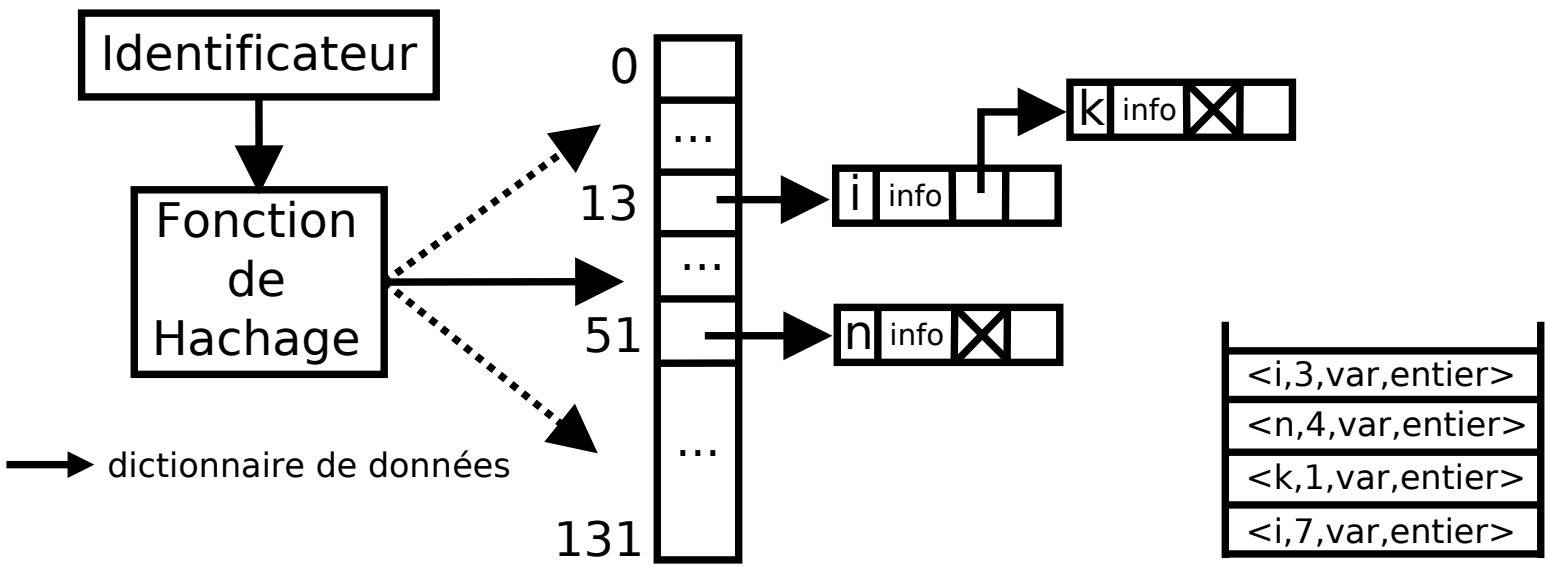


Utilisation :

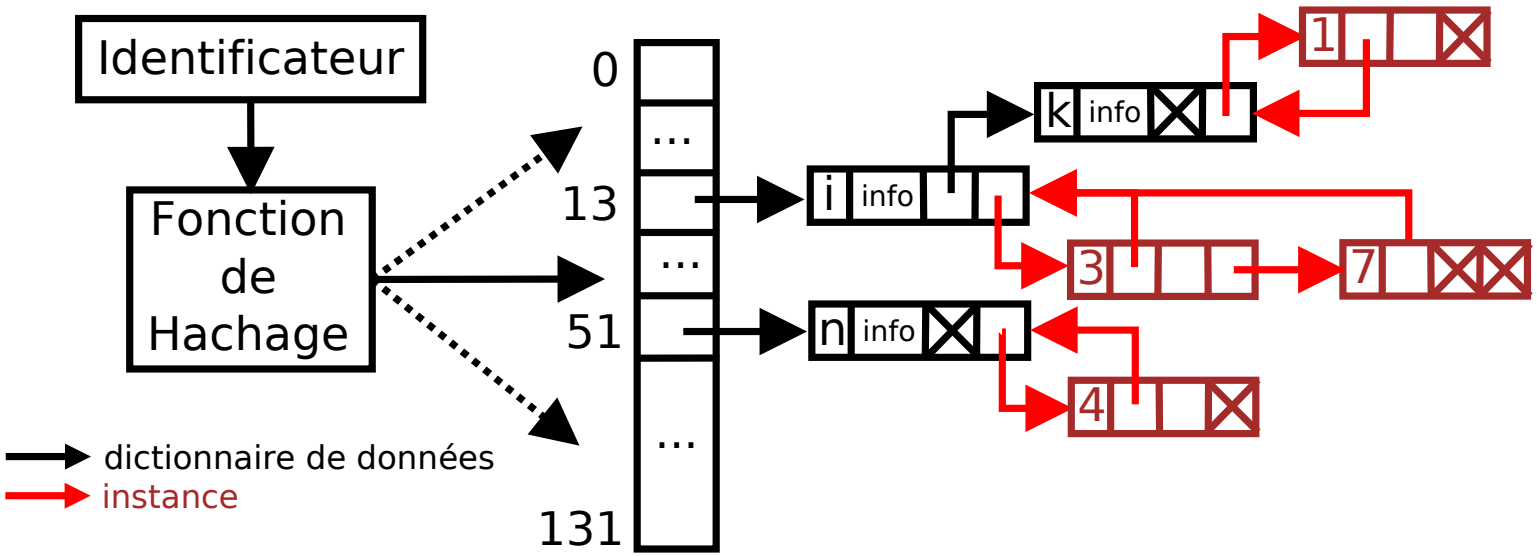
- Interpréteur(s)
- Calculs dynamiques
- Lien avec la table des symboles



Dictionnaire de données et Pile 1/3

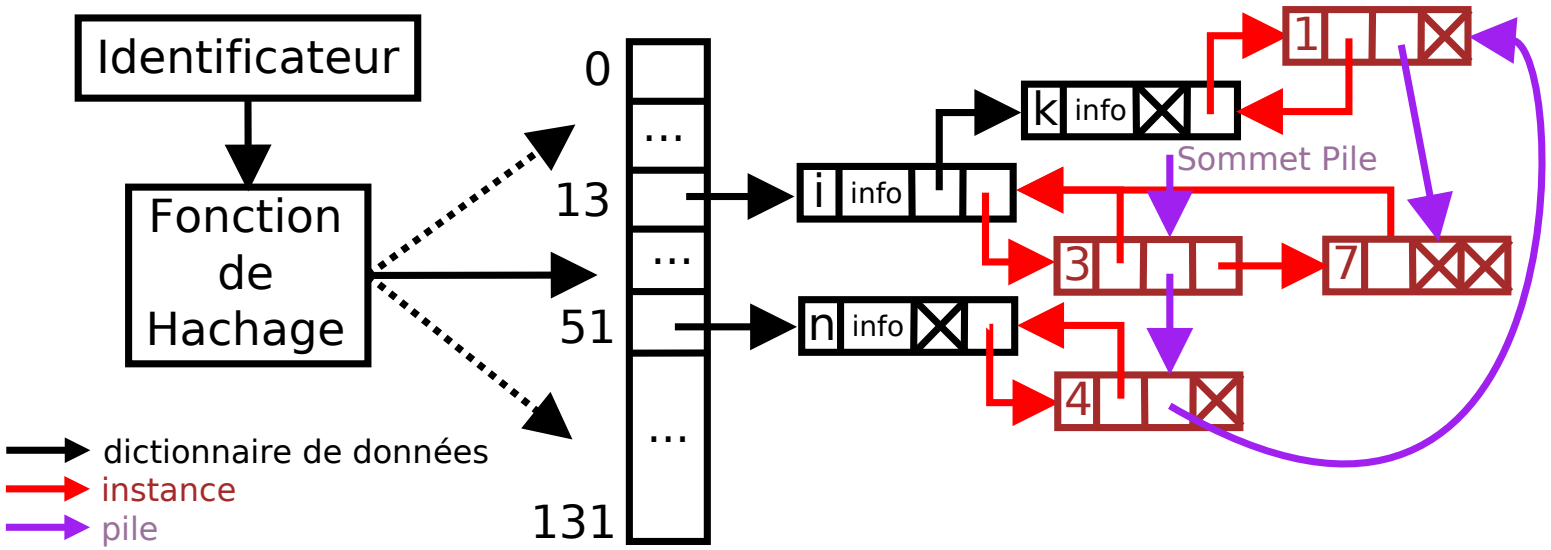


Dictionnaire de données et Pile 2/3



<i,3,var,entier>
<n,4,var,entier>
<k,1,var,entier>
<i,7,var,entier>

Dictionnaire de données et Pile 3/3



<i,3,var,entier>
<n,4,var,entier>
<k,1,var,entier>
<i,7,var,entier>

Ramasse miettes

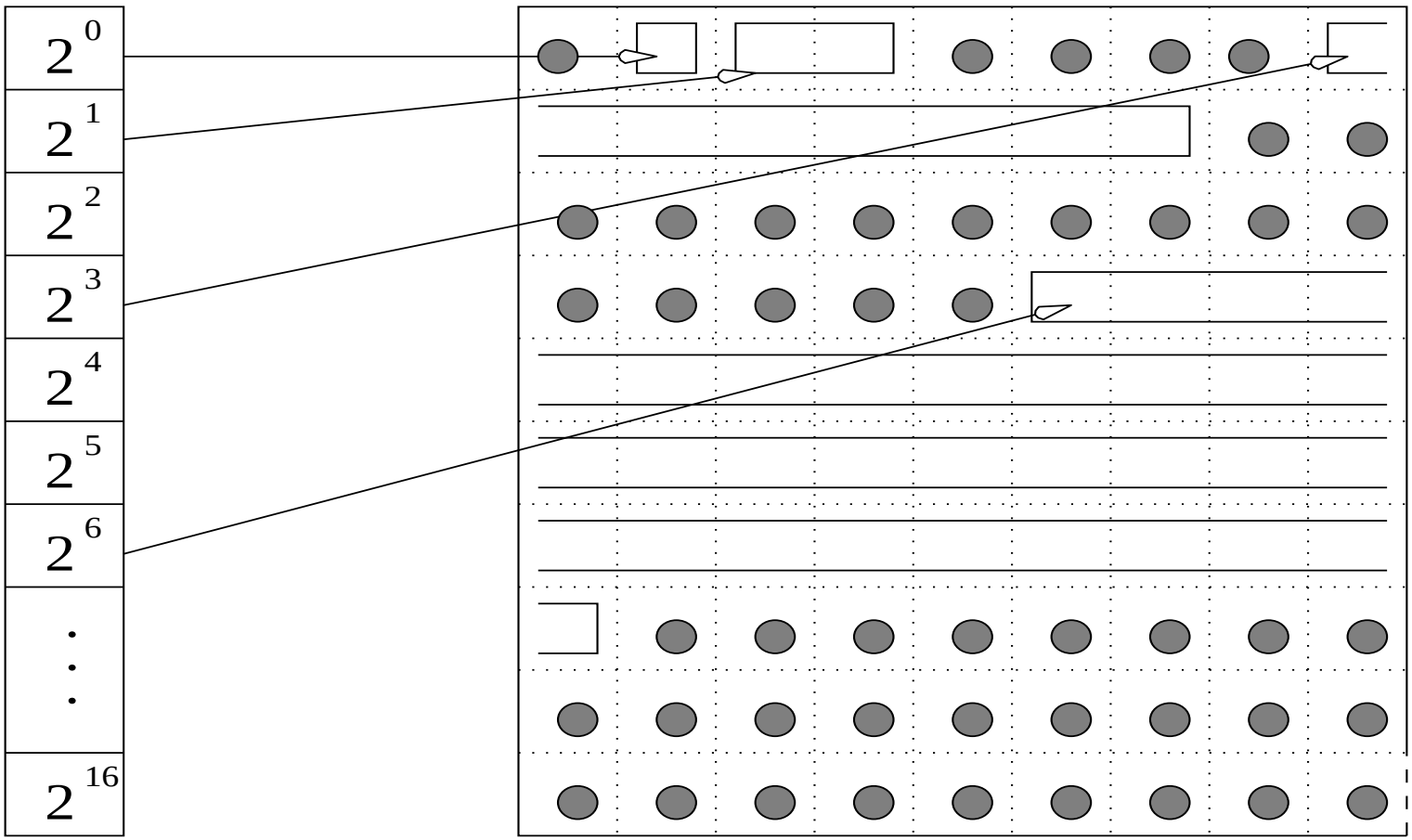
But : Gérer la mémoire d'une façon efficace
Entrée :

- un type de mémoire
- les objets manipulés

Action sur les éléments :

- Rechercher,
- Ajouter,
- Supprimer

Exemple de gestion tas



Ramasse Stratégies

But : Récupérer la mémoire libérée

Stratégies de reconstruction :

- *Rien* : le bloc mémoire est libéré mais perdu
- *Recollement* :
  - Recherche des blocs libres adjacents
  - Reconstruction d'un bloc plus gros
  - Redécoupage (éventuellement)

Pas de bloc de grande taille :

- Taille libre $<$ taille demandée
- Optimisation des blocs libres

État mémoire

Propriétés :

- Mi-chemin entre la pile et la table
- Pile pour le masquage
- Table pour l'accés direct aux attributs

Une **entrée** : $QUAD = ID \times VAL \times OBJ \times SORTE$

- ID : l'ensemble des identificateurs
- VAL : l'ensemble des valeurs associées
- OBJ : la nature de l'objet représenté
- $SORTE$: le type de l'objet

Prototypage (1)

Gestion de la pile :

Créer, [] :		$\rightarrow$ MEM
Empiler, . :	QUA $\times$ MEM	$\rightarrow$ MEM
Dépiler :	MEM	$\rightarrow$ MEM
Echanger :	MEM	$\rightarrow$ MEM

Opérations à partir des constructeurs :

DeclVar :	ID $\times$ VAL $\times$ SORTE $\times$ MEM	$\rightarrow$ MEM
IdentVal :	ID $\times$ SORTE $\times$ MEM $\times$ NAT	$\rightarrow$ MEM
DeclCst :	ID $\times$ VAL $\times$ SORTE $\times$ MEM	$\rightarrow$ MEM
DeclTab :	ID $\times$ VAL $\times$ SORTE $\times$ MEM	$\rightarrow$ MEM
DeclMeth :	ID $\times$ VAL $\times$ SORTE $\times$ MEM	$\rightarrow$ MEM

Axiomatique (1)

Axiome de Gestion de la pile :

- $Empiler(q, m) = q.m$
- $Depiler(q.m) = m$
- $Echanger(q_1.q_2.m) = q_2.q_1.m$
- $DeclVar(i, v, t, m) = < i, v, var, t > .m$
- $IdentVal(i, t, [], s) = []$
- $IdentVal(i, t, < i_1, v_1, o_1, t_1 > .m, s) =$ **Si** $s == 0$ **alors** $< i, v_1, var, t > .m$ **sinon** $< i_1, v_1, o_1, t_1 >.IdentVal(i, t, m, s - 1)$
- $DeclCst(i, v, t, m) =$ **Si** $v == w$ **alors** $< i, v, vcst, t > .m$ **sinon** $< i, v, cst, t > .m$
- $DeclTab(i, v, t, m) = < i,$ CréerTas$(v, t, m), tab, t > .m$
- $DeclMeth(i, v, t, m) = < i, v, meth, t > .m$

Prototypage (2)

Opérations de modification de la mémoire :

$$
\begin{array}{rll}
\text{AffecterVal} : & \text{ID} \times \text{VAL} \times \text{MEM} & \rightarrow \text{MEM} \\
\text{AffecterValT} : & \text{ID} \times \text{VAL} \times \text{VAL} \times \text{MEM} & \rightarrow \text{MEM} \\
\text{AffecterType} : & \text{ID} \times \text{SORTE} \times \text{MEM} & \rightarrow \text{MEM} \\
\text{RetirerDecl} : & \text{ID} \times \text{MEM} & \rightarrow \text{MEM} \\
\text{ExpParam} : & \text{VAL} \times \text{VAL} \times \text{MEM} & \rightarrow \text{MEM}
\end{array}
$$

Déclaration :

- $\mathit{RetirerDecl}(i, []) = []$
- $\mathit{RetirerDecl}(i, < i_1, v_1, o, t > .m) =$ **Si** $i == i_1$ **alors** (**Si** $o == \mathit{tab}$ **alors** RetirerTas(v_1,t) **endif** m) **sinon** $< i_1, v_1, o, t > .\text{RetirerDecl}(i, m)$

Prototypage (3)

Affectation :

- $AffecterVal(i, v, < i_1, v_1, o, t > .m) =$ **Si** $i \neq i_1$ **alors** $< i_1, v_1, o, t > .\text{AffecterVal}(i, v, m)$ **sinon Si** $o == vcst$ **alors** $< i, v, cst, t > .m$ **sinon Si** $o == cst$ **alors** m **sinon Si** $o == tab$ **alors** $\text{AjouterRef}(v, t)$, $\text{RetirerTas}(v_1, t)$ **sinon** $< i, v, o, t > .m$
- $AffecterValT(i, v, ind, < i_1, v_1, o, t > .m) =$ **Si** $i \neq i_1$ **alors** $< i_1, v_1, o, t > .\text{AffecterValT}(i, v, ind, m)$ **sinon** $\text{AffecterTas}(v_1, ind, v, m)$
- $AffecterType(i, t, < i_1, v_1, o, t_1 > .m) =$ **Si** $i == i_1$ **alors** $< i, v_1, o, t > .m$ **sinon** $< i_1, v_1, o, t_1 > .\text{AffecterType}(i, t, m)$

Paramètre :

- $ExpParam(lexp, ent, m) =$ **Si** $(lexp \neq exnil)$ **et** $(ent \neq enil)$ **alors** $\text{ExpParam}(le_1, ents, \text{DeclVar}(i, v, t, m))$ **sinon** m

  / où v est définie par $m \overset{eval}{\vdash} e \Rightarrow v$, on suppose que : $lexp = listexp(e, le_1)$ et $ent = entetes(entete(t, i), ents)$ */*

Prototypage (3)

Opérations d'accès aux informations de la mémoire :

$$
\begin{array}{rll}
\text{Val} : & \text{ID} \times \text{MEM} & \rightarrow \text{VAL} \cup \{w\} \\
\text{ValT} : & \text{ID} \times \text{VAL} \times \text{MEM} & \rightarrow \text{VAL} \cup \{w\} \\
\text{Objet} : & \text{ID} \times \text{MEM} & \rightarrow \text{OBJ} \cup \{w\} \\
\text{Sorte} : & \text{ID} \times \text{MEM} & \rightarrow \text{SORTE} \cup \{w\} \\
\text{Paramètre} : & \text{ID} \times \text{MEM} & \rightarrow \text{DECLS} \cup \{w\} \\
\text{Déclaration} : & \text{ID} \times \text{MEM} & \rightarrow \text{DECLS} \cup \{w\} \\
\text{Corps} : & \text{ID} \times \text{MEM} & \rightarrow \text{INSTRS} \cup \{w\}
\end{array}
$$

- $Val(i, []) = w$
  $Val(i, < i_1, v_1, o, t > .m) =$ **Si** $i == i_1$ **alors** v_1 **sinon** $\text{Val}(i, m)$
- $ValT(i, ind, []) = w$
  $ValT(i, ind, < i_1, v_1, o, t > .m) =$ **Si** $i == i_1$ **alors** $ValeurTas(v_1, ind, m)$ **sinon** $\text{ValT}(i, ind, m)$

Axiome (3)

- $Objet(i, []) = w$
  $Objet(i, < i_1, v_1, o, t > .m) =$ **Si** $i == i_1$ **alors** o **sinon** $\text{Objet}(i, m)$
- $Sorte(i, []) = w$
  $Sorte(i, < i_1, v_1, o, t > .m) =$ **Si** $i == i_1$ **alors** t **sinon** $\text{Sorte}(i, m)$
- $Paramètre(i, []) = w$
  $Paramètre(i, < i_1, v_1, o, t > .m) =$ **Si** $i \neq i_1$ **alors** $\text{Paramètre}(i, m)$ **sinon**
  Si $o \neq meth$ **alors** w **sinon**
  Si $v_1 == \text{méthode}(t, i, ents, dv, is)$ **alors** $ents$ **sinon** w
- $Déclaration(i, []) = w$
  $Déclaration(i, < i_1, v_1, o, t > .m) =$ **Si** $i \neq i_1$ **alors** $\text{Déclaration}(i, m)$ **sinon Si**
  $o \neq meth$ **alors** w **sinon**
  Si $v_1 == \text{méthode}(t, i, ents, dvs, iss)$ **alors** dvs **sinon** w
- $Corps(i, []) = w$
  $Corps(i, < i_1, v_1, o, t > .m) =$ **Si** $i \neq i_1$ **alors** $\text{Corps}(i, m)$ **sinon**
  Si $o \neq meth$ **alors** w **sinon**
  Si $v_1 == \text{méthode}(t, i, ents, dvs, iss)$ **alors** iss **sinon** w

Syntace concrète – Exemple

1er programme

```
class C {
  int x = 0;

  int f(int p) {
    int c = 6;
    return p+c;
  };

  main {
    int y = 2;
    x = 3;
    y += f(x);
  }
}
```

Arbre d'analyse syntaxique

Syntaxe abstraite MiniJaja

classe :	ID × DECLS × MAIN	→ CLASSE
ident :	string	→ ID
decls :	DECL × DECLS	→ DECLS
vars :	DECL × DECLS	→ DECLS
vnil :		→ DECLS
cst :	TYPE × ID × EXP	→ DECL
var :	TYPE × ID × EXP	→ DECL
tableau :	TYPE × ID × EXP'	→ DECL
entêtes :	ENTETE × ENTETES	→ ENTETES
omega :		→ EXP'
entête :	TYPE × ID	→ ENTETE
enil :		→ ENTETES
main :	DECLS × INSTRS	→ MAIN
méthode :	TYPEMETH × ID × ENTETES × DECLS × INSTRS	→ DECL
rien, entier, booléen :		→ TYPEMETH
entier, booléen :		→ TYPE

Syntaxe abstraite (suite)

instrs :	INSTR × INSTRS	→ INSTRS
inil :		→ INSTRS
somme, affectation :	ID × EXP	→ INSTR
incrément :	ID	→ INSTR
appelI :	ID × LISTEXP	→ INSTR
retour :	EXP	→ INSTR
si :	EXP × INSTRS × INSTRS	→ INSTR
tantque :	EXP × INSTRS	→ INSTR
listexp :	EXP × LISTEXP	→ LISTEXP
exnil :		→ LISTEXP
appelE :	ID × LISTEXP	→ EXP'
tab :	ID × EXP	→ ID
non,moins :	EXP	→ EXP'
ou, et, =, > :	EXP × EXP	→ EXP'
*, /, +, - :	EXP × EXP	→ EXP'
nbre :	integer	→ EXP'
vrai, faux :		→ EXP'

Syntaxe abstraite

La **syntaxe abstraite** consiste à définir :

- un ensemble de noms de fonction dont on donne le profil
- un ensemble de types de fonctions permettant de typer les fonctions.

Syntaxe abstraite à partir de la **syntaxe concrète** :

- à chaque règle de la syntaxe concrète est associée un opérateur
- à chaque non terminal de la syntaxe concrète est associé un type d'opérateur

MEM $\vdash$ INSTR $\rightarrow$ MEM

- [classe] : $$\frac{\begin{array}{l}\text{DeclVar}(i, w, w, []) \vdash dss \rightarrow m_1 \\ m_1 \vdash mma \rightarrow m_2 \\ m_2 \overset{retrait}{\vdash} dss \rightarrow m_3\end{array}}{[] \vdash \text{classe}(\text{ident}(i), dss, mma) \rightarrow RetirerDecl(ident(i), m_3)}$$
- [tableau] : $$\frac{m \overset{eval}{\vdash} e \Rightarrow v}{m \vdash \text{tableau}(t, ident(i), e) \rightarrow \text{DeclTab}(i, v, t, m)}$$
- [decls] : $$\frac{m \vdash ds \rightarrow m_1,\ m_1 \vdash dss \rightarrow m_2}{m \vdash \text{decls}(ds, dss) \rightarrow m_2}$$
- [vars] : $$\frac{m \vdash dv \rightarrow m_1,\ m_1 \vdash dvs \rightarrow m_2}{m \vdash \text{vars}(dv, dvs) \rightarrow m_2}$$

S. Interprétative (Suite...)

- [main] : $$\frac{\begin{array}{l} m \vdash dvs \rightarrow m_1 \\ m_1 \vdash iss \rightarrow m_2 \end{array} \quad m_2 \overset{retrait}{\vdash} dvs \rightarrow m_3}{m \vdash \text{main}(dvs, iss) \rightarrow m_3}$$

- [cst] : $$\frac{m \overset{eval}{\vdash} e \Rightarrow v}{m \vdash \text{cst}(t, ident(i), e) \rightarrow \text{DeclCst}(i, v, t, m)}$$

- [var] : $$\frac{m \overset{eval}{\vdash} e \Rightarrow v}{m \vdash \text{var}(t, ident(i), e) \rightarrow \text{DeclVar}(i, v, t, m)}$$

- [vnil] : $m \vdash \text{vnil} \rightarrow m$

- [méthode] : $m \vdash \text{méthode}(t, ident(i), ent, dvs, iss) \rightarrow \text{DeclMeth}(i, \text{méthode}(t, ident(i), ent, dvs, iss), \text{t}, m)$

S. Interprétative (...Suite...)

- [instrs] : $$\frac{m \vdash is \rightarrow m_1, m_1 \vdash iss \rightarrow m_2}{m \vdash \text{instrs}(is, iss) \rightarrow m_2}$$
- [inil] : $m \vdash inil \rightarrow m$
- [affectation] : $$\frac{m \overset{eval}{\vdash} e \Rightarrow v}{m \vdash \text{affectation}(ident(i), e) \rightarrow \text{AffecterVal}(i, v, m)}$$
- [affectationT] : $$\frac{m \overset{eval}{\vdash} e \Rightarrow v,\ m \overset{eval}{\vdash} e_1 \Rightarrow ind}{m \vdash \text{affectation}(\text{tab}(\text{ident}(i), e_1), e) \rightarrow \text{AffecterValT}(i, ind, v, m)}$$
- [incrément] : $m \vdash \text{incrément}(ident(i)) \rightarrow \text{AffecterVal}(i, Val(i, m) + 1, m)$
- [incrémentT] : $$\frac{m \overset{eval}{\vdash} e \Rightarrow ind}{m \vdash \text{incrément}(\text{tab}(\text{ident}((i), e)) \rightarrow \text{AffecterValT}(i, ind, ValT(i, ind, m) + 1, m)}$$

S. Interprétative (...Suite...)

- $$[\text{appelI}] : \frac{\begin{array}{l}\text{ExpParam}(lexp, \text{Paramètre}(i, m), m) \vdash \text{Déclaration}(i, m) \rightarrow m_1 \\ m_1 \vdash \text{Corps}(i, m) \rightarrow m_2 \\ m_2 \overset{retrait}{\vdash} \text{Déclaration}(i, m) \rightarrow m_3 \\ m_3 \overset{retrait}{\vdash} \text{Paramètre}(i, m) \rightarrow m_4\end{array}}{m \vdash \text{appelI}(ident(i), lexp) \rightarrow m_4}$$

- $$[\text{retour}] : \frac{m \overset{eval}{\vdash} e \Rightarrow v}{m \vdash \text{retour}(e) \rightarrow \text{AffecterVal}(\text{VariableClasse(m)}, v, m)}$$

- $$[\text{somme}] : \frac{m \overset{eval}{\vdash} e \Rightarrow v}{m \vdash \text{somme}(\text{ident}(i), e) \rightarrow \text{AffecterVal}(i, Val(i, m) + v, m)}$$

- $$[\text{sommeT}] : \frac{m \overset{eval}{\vdash} e \Rightarrow v,\ m \overset{eval}{\vdash} e_1 \Rightarrow ind}{\begin{array}{l}m \vdash \text{somme}(\text{tab}(\text{ident}(i), e_1), e) \\ \qquad \rightarrow \text{AffecterValT}(i, ind, ValT(i, ind, m) + v, m)\end{array}}$$

S. Interprétative (...Suite)

- [sivrai] : $$\frac{m \overset{eval}{\vdash} e \Rightarrow true,\ m \vdash iss \rightarrow m_1}{m \vdash \text{si}(e,\ iss,\ iss_1) \rightarrow m_1}$$
- [sifaux] : $$\frac{m \overset{eval}{\vdash} e \Rightarrow false,\ m \vdash iss_1 \rightarrow m_1}{m \vdash \text{si}(e,\ iss,\ iss_1) \rightarrow m_1}$$
- [tantquevrai] : $$\frac{\begin{array}{l} m \overset{eval}{\vdash} e \Rightarrow true \\ m \vdash iss \rightarrow m_1 \\ m_1 \vdash \text{tantque}(e, iss) \rightarrow m_2 \end{array}}{m \vdash \text{tantque}(e,\ iss) \rightarrow m_2}$$
- [tantquefaux] : $$\frac{m \overset{eval}{\vdash} e \Rightarrow false}{m \vdash \text{tantque}(e, iss) \rightarrow m}$$

$MEM \overset{eval}{\vdash} EXP \Rightarrow VAL$

- [tab] : $\dfrac{m \overset{eval}{\vdash} e \Rightarrow v}{m \overset{eval}{\vdash} \text{tab(ident}(i), e) \Rightarrow ValT(i, v, m)}$
- [non] : $\dfrac{m \overset{eval}{\vdash} e \Rightarrow v}{m \overset{eval}{\vdash} \text{non}(e) \Rightarrow \neg v}$
- [moins] : $\dfrac{m \overset{eval}{\vdash} e \Rightarrow v}{m \overset{eval}{\vdash} \text{moins}(e) \Rightarrow -v}$
- [op2] : $\dfrac{m \overset{eval}{\vdash} e \Rightarrow v,\ m \overset{eval}{\vdash} e_1 \Rightarrow v_1}{m \overset{eval}{\vdash} \text{op2}(e, e_1) \Rightarrow v\ op\ v_1}$
- [ident] : $m \overset{eval}{\vdash} \text{ident(i)} \Rightarrow \text{Val(i,m)}$
- [vrai] : $m \overset{eval}{\vdash} \text{vrai} \Rightarrow \text{true}$
- [faux] : $m \overset{eval}{\vdash} \text{faux} \Rightarrow \text{false}$
- [nbre] : $m \overset{eval}{\vdash} \text{nbre(n)} \Rightarrow \text{n}$
- [omega] : $m \overset{eval}{\vdash} \text{omega} \Rightarrow w$
- [appelE] : $\dfrac{m \vdash \text{appelI}(\text{ident}(i), lexp) \rightarrow m_1 \qquad m_1 \overset{eval}{\vdash} \text{VariableClasse}(m_1) \Rightarrow v}{m \overset{eval}{\vdash} \text{appelE}(\text{ident}(i), lexp) \Rightarrow \text{v}}$

$MEM \overset{retrait}{\vdash} DECLS \cup ENTETES \cup ENTETE \rightarrow MEM$

- [rdecls] : $$\frac{m \overset{retrait}{\vdash} dss \rightarrow m_1,\ m_1 \overset{retrait}{\vdash} ds \rightarrow m_2}{m \overset{retrait}{\vdash} \text{decls}(ds, dss) \rightarrow m_2}$$
- [rentêtes],[rvars] : $$\frac{m \overset{retrait}{\vdash} dvs \rightarrow m_1,\ m_1 \overset{retrait}{\vdash} dv \rightarrow m_2}{m \overset{retrait}{\vdash} \text{vars}(dv, dvs) \rightarrow m_2}$$
- [renil],[rvnil] : $m \overset{retrait}{\vdash} \text{vnil} \rightarrow m$
- [rcst],[rentête],[rvar] : $m \overset{retrait}{\vdash} \text{var}(t, ident(i), e) \rightarrow \text{RetirerDecl}(i, m)$
- [rtableau] : $m \overset{retrait}{\vdash} \text{tableau}(t, ident(i), e) \rightarrow \text{RetirerDecl}(i, m)$
- [rméthode] : $m \overset{retrait}{\vdash} \text{méthode}(t, \text{ident}(i), en, dvs, ins) \rightarrow \text{RetirerDecl}(i, m)$

Jaja-Code

C'est un **langage** associé à une machine virtuelle :

- d'instructions (Jaja-Code)
- d'une pile
- d'un tas (Objet)

La *pile* est une **mémoire** contenant :

- des associations identificateurs / valeurs
- des valeurs résultats
- des adresses d'instructions
- des valeurs effectives

Les instructions Jaja-Code

- init
- new(ident,type,sorte,val)
- invoke(ident)
- Tpush(valeur)
- Tload(ident)
- goto(adresse)
- Tinc(ident)
- Tadd, Tmul, Tsub, Tdiv, Tor
- Tand, Tneg, Tcmp, Tsup
- Tswap
- newarray(ident,type)
- Treturn
- Tpop
- Tstore(ident)
- if(adresse)
- nop

Syntaxe concrète du Jaja-Code

classe	: :=	adresse instrs ; classe |	*JajaCode($2,$4)*
		vide	*jcnil*
instrs	: :=	init | swap |	*init* | *swap*
		new(ident, type, sorte, adr) |	*new($3,$5,$7,$9)*
		newarray(ident, type) |	*newarray($3,$5)*
		invoke(ident) |	*invoke($3)*
		return |	*return*
		push(valeur) | pop |	*push($3)* | *pop*
		load(ident) | aload(ident) |	*load($3)* | *aload($3)*
		store(ident) | astore(ident) |	*store($3)* | *astore($3)*
		if(adresse) | goto(adresse) |	*if($3)* | *goto($3)*

Syntaxe concrète (Suite)

instrs	::=	inc(ident) |ainc(ident) |	*inc($3) | ainc($3)*
		oper | nop |	*oper | nop*
		jcstop	*jcstop*
ident	::=	IDENTIFIER	*jcident($1)*
valeur	::=	NOMBRE |	*jcnbre($1)*
		vrai |faux |	*jcvrai |jcfaux*
		vide	*jcnil*
adresse	::=	NOMBRE	*jcnbre($1)*
oper	::=	oper2 | oper1	$1 | $1
oper1	::=	neg | not	*neg | not*
oper2	::=	add | sub | mul | div | cmp | sup| or | and	*oper2*

Syntaxe abstraite du Jaja-Code

JajaCode :	JCODE × JCODES	→ JCODES
jcnil :		→ JCODES
init, pop, nop, jcstop, return, swap, oper :		→ JCODE
invoke, inc, ainc :	ID	→ JCODE
load, store, aload, astore :	ID	→ JCODE
new :	ID × TYPE × SORTE × JCVAL	→ JCODE
newarray :	ID × TYPE	→ JCODE
push :	JCVAL	→ JCODE
goto, if :	ADR	→ JCODE
jcident :	string	→ ID
jcnbre :	entier	→ JCVAL
jcvrai, jcfaux, jc*w* :		→ JCVAL

oper2 = $\{add, sub, mul, div, cmp, sup, and, or\}$, oper1 = $\{neg, not\}$,

oper = oper1 ∪ oper2, ADR = $\mathbb{N}$

Sémantique des instructions

$$< MEM, ADR > \vdash JCODE \twoheadrightarrow < MEM, ADR >$$

- [init] : <m,a> ⊢ init ↠ < [], a+1>
- [jcstop] : <m,a> ⊢ jcstop ↠ <m, ⊥>
- [nop] : <m,a> ⊢ nop ↠ <m, a+1>
- [swap] : $< q_1.q_2$.m,a> ⊢ swap ↠ $< q_2.q_1$.m, a+1>
- [newV] : <m,a> ⊢ new(i,t,var,s) ↠ <IdentVal(i,t,m,s),a+1>
- [newC] : $<< w$,v, cst,w>.m,a> ⊢ new(i,t,cst,0) ↠ <DeclCst(i,v,t,m),a+1>
- [newM] : $<< w$, v, cst,w>.m,a> ⊢ new(i,t,meth,0) ↠ <DeclMeth(i, v, t, m),a+1>
- [newarray] : $<< w$, v, cst,w>.m,a>⊢ newarray(i, t) ↠<DeclTab(i, v, t, m), a+1>

S. instructions Jaja-Code

- [invoke] : $<m,a> \vdash invoke(i) \twoheadrightarrow << w, a+1,cst,w>.m, Val(i, m)>$
- [return] : $<< w, a_1,cst,w>.m,a> \vdash return \twoheadrightarrow <m, a_1 >$
- [goto] : $<m,a> \vdash goto(a_1) \twoheadrightarrow <m, a_1 >$
- [load] : $<m,a> \vdash load(i) \twoheadrightarrow << w, Val(i,m), cst,w>.m, a+1>$
- [aload] : $<< w, ind, cst, w >.m,a> \vdash aload(i) \twoheadrightarrow << w, ValT(i, ind, m), cst,w>.m, a+1>$
- [store] : $<< w, v, cst, w >.m,a> \vdash store(i) \twoheadrightarrow <AffecterVal(i,v,m), a+1>$
- [astore] : $<< w, v, cst, w > . < w, ind, cst, w >.m,a> \vdash astore(i) \twoheadrightarrow < AffecterValT(i, ind, v, m), a+1>$

S. instructions Jaja-Code

- [push] : <m,a> $\vdash$ push(v) $\twoheadrightarrow$ << w, v, cst,w>.m, a+1>
- [pop] : <q.m,a> $\vdash$ pop $\twoheadrightarrow$ < m, a+1>
- [iftrue] : << w,true, cst, w>.m,a> $\vdash$ if(a_1)$\twoheadrightarrow$ <m,a_1 >
- [iffalse] : << w,false, cst, w>.m,a> $\vdash$ if(a_1)$\twoheadrightarrow$ <m,a+1>
- [op2] : << w, v_2,cst,w>.< w, v_1,cst,w>.m,a> $\vdash$ oper2 $\twoheadrightarrow$ << w, v_1 oper2 v_2, cst,w>.m, a+1>
- [op1] : << w, v_1,cst,w>.m,a> $\vdash$ oper1 $\twoheadrightarrow$ << w, oper1 v_1, cst,w>.m, a+1>
- [inc] : << w,v, cst,w>.m,a>$\vdash$ inc(i) $\twoheadrightarrow$ <AffecterVal(i,Val(i,m)+v,m), a+1>
- [ainc] : << w, v, cst, w > . < w, *ind*, cst, w >.m,a> $\vdash$ ainc(i) $\twoheadrightarrow$ <AffecterValT(i,ind,ValT(i,ind,m)+v,m), a+1>

Comp. MiniJaja → Jaja-Code

Compilateur :

- A tout programme MiniJaja associe un programme Jaja-Code
- Assurer "l'équivalence" sémantique

Opérateur :

- $\oplus$: JCODES $\times$ JCODES $\rightarrow$ JCODES
- $\oplus_G$: JCODE $\times$ JCODES $\rightarrow$ JCODES
- $\oplus_D$: JCODES $\times$ JCODE $\rightarrow$ JCODES

Liste neutre : jcnil

Compilation Déclarations

- $$[\text{cclasse}] : \frac{\begin{array}{rcll} n+1 & \vdash & dss & \Rightarrow \{pdss,\ ndss\} \\ n+ndss+1 & \vdash & mma & \Rightarrow \{pmma, nmma\} \\ n+ndss+nmma+1 & \vdash^{retrait} & dss & \Rightarrow \{prdss,\ nrdss\} \end{array}}{\begin{array}{l} n \vdash \text{classe}(i,\ dss,\ mma) \Rightarrow \{init \oplus_G (pdss \oplus pmma \oplus prdss) \\ \quad \oplus_D pop \oplus_D jcstop,\ ndss+nmma+nrdss+3\} \end{array}}$$

- $$[\text{cdecls}] : \frac{n \vdash ds \Rightarrow \{pds,\ nds\},\ n+nds \vdash dss \Rightarrow \{pdss,\ ndss\}}{n \vdash \text{decls}(ds,\ dss) \Rightarrow \{pds \oplus pdss,\ nds+ndss\}}$$

- $$[\text{cvars}] : \frac{n \vdash dv \Rightarrow \{pdv,\ ndv\},\ n+ndv \vdash dvs \Rightarrow \{pdvs,\ ndvs\}}{n \vdash \text{vars}(dv,\ dvs) \Rightarrow \{pdv \oplus pdvs,\ ndv+ndvs\}}$$

- $$[\text{cvnil}] : n \vdash vnil \Rightarrow \{jcnil,\ 0\}$$

- $$[\text{cvar}] : \frac{n \vdash e \Rightarrow \{pe, ne\}}{n \vdash \text{var}(t, \text{ident}(i), e) \Rightarrow \{pe \oplus_D \text{new}(i, t, var, 0),\ ne+1\}}$$

Compilation Déclarations (suite)

- [ccst] : $$\frac{n \vdash e \Rightarrow \{pe, ne\}}{n \vdash \text{cst}(t, \text{ident}(i), e) \Rightarrow \{pe \oplus_D \text{new}(i, t, cst, 0),\ ne + 1\}}$$
- [ctableau] : $$\frac{n \vdash e \Rightarrow \{pe, ne\}}{n \vdash \text{tableau}(t, \text{ident}(i), e) \Rightarrow \{pe \oplus_D \text{newarray}(i, t), ne + 1\}}$$
- [centêtes] : $$\frac{\begin{array}{rcl} n & \vdash & en \Rightarrow \{pen,\ nen\}, \\ n + nen & \vdash & ens \Rightarrow \{pens,\ nens\} \end{array}}{n \vdash \text{entêtes}(en,\ ens) \Rightarrow \{pen \oplus pens, nen + nens\}}$$
- [centête] : $n \vdash \text{entête}(t, i) \Rightarrow \{\text{new}(i, t, var, k) \oplus_G jcnil,\ 1\}$
- [cmain] : $$\frac{\begin{array}{rcl} n & \vdash & dv \Rightarrow \{pdvs, ndvs\} \\ n + ndvs & \vdash & iss \Rightarrow \{piss, niss\} \\ n + ndvs + niss + 1 & \overset{retrait}{\vdash} & dvs \Rightarrow \{prdvs, nrdvs\} \end{array}}{\begin{array}{l} n \vdash \text{main}(dvs, iss) \Rightarrow \{pdvs \oplus piss \oplus (push(0) \oplus_G prdvs), \\ \qquad ndvs + niss + nrds + 1\} \end{array}}$$

Compilation déclaration

$$[\text{cméthode}] : \frac{\begin{array}{rcll} n+3 & \vdash & ens & \Rightarrow \{pens,\ nens\} \\ n+nens+3 & \vdash & dvs & \Rightarrow \{pdvs,\ ndvs\} \\ n+nens+ndvs+3 & \vdash & iss & \Rightarrow \{piss,\ niss\} \\ n+nens+ndvs+niss+3 & \overset{retrait}{\vdash} & dvs & \Rightarrow \{prdvs,\ nrdvs\} \end{array}}{\begin{array}{l} n \vdash \text{méthode}(t,i,ens,dvs,iss) \Rightarrow \{jcnil \oplus_D push(n+3) \\ \quad \oplus_D new(i,t,meth,0) \oplus_D goto(n+nens+ndvs+niss+nrdvs+5) \\ \quad \oplus pens \oplus pdvs \oplus piss \oplus prdvs \oplus_D swap \oplus_D return, \\ \quad nens+ndvs+niss+nrdvs+5\} \end{array}}$$

$$[\text{cméthodeR}] : \frac{\vdots}{\begin{array}{l} n \vdash \text{méthode}(t,i,en,d,s) \Rightarrow \{jcnil \oplus_D push(n+3) \\ \oplus_D new(i,t,meth,0) \oplus_D goto(n+nens+ndvs+niss+nrdvs+6) \\ \oplus pens \oplus pdvs \oplus piss \oplus (push(0) \oplus_G prdvs) \oplus_D swap \oplus_D return, \\ nens+ndvs+niss+nrdvs+6\} \end{array}}$$

Compilation Instructions

- [cinstrs] : $$\frac{n \vdash is \Rightarrow \{ps,\ nis\},\ n + nis \vdash iss \Rightarrow \{piss, niss\}}{n \vdash instrs(is, iss) \Rightarrow \{pis \oplus piss,\ nis + niss\}}$$
- [csomme] : $$\frac{n \vdash e \Rightarrow \{pe,\ ne\}}{n \vdash \text{somme}(ident(i), e) \Rightarrow \{pe \oplus_D inc(i),\ ne + 1\}}$$
- [csommeT] : $$\frac{\begin{array}{rl} n \vdash & e \Rightarrow \{pe,\ ne\} \\ n + ne \vdash & e_1 \Rightarrow \{pe_1,\ ne_1\} \end{array}}{\begin{array}{l} n \vdash \text{somme}(\text{tab}(\text{ident}(i), e), e_1) \Rightarrow \\ \quad \{pe \oplus pe_1 \oplus_D ainc(i), ne + ne_1 + 1\} \end{array}}$$
- [cinc] : $n \vdash \text{incrément}(ident(i)) \Rightarrow \{jcnil \oplus_D push(1) \oplus_D inc(i), 2\}$
- [cincT] : $$\frac{n \vdash e \Rightarrow \{pe,\ ne\}}{\begin{array}{l} n \vdash \text{incrément}(\text{tab}(\text{ident}(i), e)) \Rightarrow \\ \quad \{pe \oplus_D push(1) \oplus_D ainc(i), ne + 2\} \end{array}}$$

Comp. Instructions (suite...)

- [caffecte] : $$\frac{n \vdash e \Rightarrow \{pe,\ ne\}}{n \vdash \text{affectation}(ident(i), e) \Rightarrow \{pe \oplus_D store(i), ne + 1\}}$$

- [caffecteT] : $$\frac{\begin{array}{r} n \vdash e \Rightarrow \{pe,\ ne\} \\ n + ne \vdash e_1 \Rightarrow \{pe_1,\ ne_1\} \end{array}}{\begin{array}{l} n \vdash \text{affectation}(\text{tab}(\text{ident}(i), e), e_1) \Rightarrow \\ \qquad \{pe \oplus pe_1 \oplus_D astore(i), ne + ne_1 + 1\} \end{array}}$$

- [cappelE] : $$\frac{\begin{array}{r} n \vdash lexp \Rightarrow \{plexp,\ nlexp\} \\ n + nlexp + 1 \underset{retrait}{\vdash} lexp \Rightarrow \{prlexp,\ nrlexp\} \end{array}}{\begin{array}{l} n \vdash \text{appelE}(\text{ident}(i), lexp) \Rightarrow \{plexp \oplus_D invoke(i) \\ \qquad \oplus\ prlexp, nlexp + nrlexp + 1\} \end{array}}$$

- [cappelI] : $$\frac{\begin{array}{r} n \vdash lexp \Rightarrow \{plexp,\ nlexp\} \\ n + nlexp + 1 \underset{retrait}{\vdash} lexp \Rightarrow \{prlexp,\ nrlexp\} \end{array}}{\begin{array}{l} n \vdash \text{appelI}(\text{ident}(i), lexp) \Rightarrow \{plexp \oplus_D invoke(i) \\ \qquad \oplus\ prlexp \oplus_D pop, nlexp + nrlexp + 2\} \end{array}}$$

Comp. Instructions (...suite)

- [cretour] : $$\frac{n \vdash e \Rightarrow \{pe,\ ne\}}{n \vdash \text{retour}(e) \Rightarrow \{pe,\ ne\}}$$

- [csi] : $$\frac{\begin{array}{rcl} n & \vdash & e \Rightarrow \{pe,\ ne\} \\ n+ne+1 & \vdash & s_1 \Rightarrow \{ps_1,\ ns_1\} \\ n+ne+ns_1+2 & \vdash & s \Rightarrow \{ps, ns\} \end{array}}{\begin{array}{l} n \vdash \text{si}(e,\ s,\ s_1) \Rightarrow \{(pe \oplus_D if(n+ne+ns_1+2)) \oplus \\ \quad ps_1 \oplus_D goto(n+ne+ns_1+ns+2) \oplus ps, \\ \quad ne+ns_1+ns+2\} \end{array}}$$

- [ctantque] : $$\frac{\begin{array}{rcl} n & \vdash & e \Rightarrow \{pe,\ ne\} \\ n+ne+2 & \vdash & iss \Rightarrow \{piss, niss\} \end{array}}{\begin{array}{l} n \vdash \text{tantque}(e, iss) \Rightarrow \{(pe \oplus_D not) \oplus_D \\ if(n+ne+niss+3) \oplus piss \oplus_D goto(n), ne+niss+3\} \end{array}}$$

Compilation Expressions

- [ctab] : $$\frac{n \vdash e \Rightarrow \{pe,\ ne\}}{n \vdash tab(ident(i), e) \Rightarrow \{pe \oplus aload(i),\ ne+1\}}$$
- [cident] : $n \vdash ident(i) \Rightarrow \{jcnil \oplus_D load(i),\ 1\}$
- [cnbre] : $n \vdash nbre(n) \Rightarrow \{jcnil \oplus_D push(n),\ 1\}$
- [cvrai] : $n \vdash vrai \Rightarrow \{jcnil \oplus_D push(jcvrai),\ 1\}$
- [cfaux] : $n \vdash faux \Rightarrow \{jcnil \oplus_D push(jcfaux),\ 1\}$
- [cop1] : $$\frac{n \vdash e \Rightarrow \{pe,\ ne\}}{n \vdash \mathrm{op1}(e) \Rightarrow \{pe \oplus_D op1,\ ne+1\}}$$
- [cop2] : $$\frac{\begin{array}{rl} n & \vdash e_1 \Rightarrow \{pe_1,\ ne_1\} \\ n+ne_1 & \vdash e_2 \Rightarrow \{pe_2,\ ne_2\} \end{array}}{n \vdash \mathrm{op2}(e_1, e_2) \Rightarrow \{(pe_1 \oplus pe_2) \oplus_D op2,\ ne_1+ne_2+1\}}$$
- [clexp] : $$\frac{\begin{array}{rl} n & \vdash exp \Rightarrow \{pexp,\ nexp\} \\ n+nexp & \vdash lexp \Rightarrow \{plexp,\ nlexp\} \end{array}}{n \vdash \mathrm{listexp}(exp, lexp) \Rightarrow \{(pexp \oplus plexp), nexp+nlexp\}}$$

Comp. Retrait des déclarations

- [crexnil],[renil],[rvnil] : $n \overset{r}{\vdash} \text{vnil} \Rightarrow \{jcnil, 0\}$
- [crvar] : $n \overset{r}{\vdash} \text{var}(t, \text{ident}(i), e) \Rightarrow \{jcnil \ \oplus_D swap \oplus_D pop, 2\}$
- [crcst] : $n \overset{r}{\vdash} \text{cst}(t, \text{ident}(i), e) \Rightarrow \{jcnil \ \oplus_D swap \oplus_D pop, 2\}$
- [crméthode] : $n \overset{r}{\vdash} \text{méthode}(t, \text{ident}(i), en, d, s) \Rightarrow \{jcnil \oplus_D swap \oplus_D pop, 2\}$
- [crdecls] : $$\frac{n \overset{r}{\vdash} dss \Rightarrow \{prdss, nrdss\}, n + nrdss \overset{r}{\vdash} ds \Rightarrow \{prds, nrds\}}{n \overset{r}{\vdash} decls(ds, dss) \Rightarrow \{prdss \oplus prds, nrdss + nrds\}}$$
- [crvars] : $$\frac{n \overset{r}{\vdash} dvs \Rightarrow \{prdvs, nrdvs\}, n + nrdvs \overset{r}{\vdash} dv \Rightarrow \{prdv, nrdv\}}{n \overset{r}{\vdash} \text{vars}(dv, dvs) \Rightarrow \{prdvs \oplus prdv, nrdvs + nrdv\}}$$
- [crlexp] : $$\frac{n \overset{r}{\vdash} lexp \ \Rightarrow \ \{prlexp, nrlexp\}}{n \overset{r}{\vdash} \text{listexp}(e, lexp) \Rightarrow \{jcnil \oplus_D swap \oplus_D pop \oplus prlexp, nrlexp + 2\}}$$

Exemple

Code MiniJaja	Adresse	JajaCode
class C {	1	init
int x = 0 ;	2	push(0)
	3	new(x,entier,var,0)
int f(**int** p) {	4	push(7)
	5	new(f,entier,meth,0)
	6	goto(17)
	7	new(p,entier,var,1)
int c = 6 ;	8	push(6)
	9	new(c,entier,var,0)
return p+c ;	10	load(p)
	11	load(c)
	12	add // *val. résultat en sommet de pile*

Exemple (suite...)

Code MiniJaja	Adresse	JajaCode
}	13	swap // *retrait de c*
	14	pop
	15	swap // *adr retour* $\leftrightarrow$ *val. résultat*
	16	return
main {		
int y = 2 ;	17	push(2)
	18	new(y,entier,var,0)
x = 3 ;	19	push(3)
	20	store(x)
y += f(x) ;	21	load(x)
	22	invoke(f)
	23	swap // *retrait du paramètre*
	24	pop
	25	inc(y)

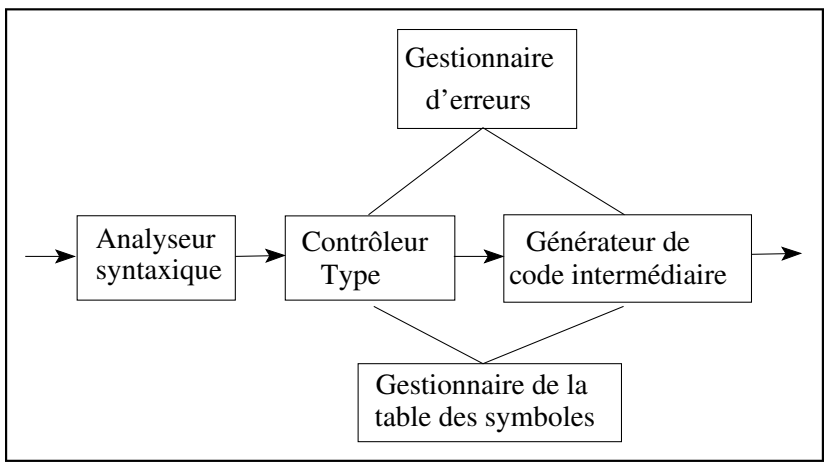
Exemple (...suite)

Code MiniJaja	Adresse	JajaCode
}	26	push(0) *//valeur pour les méthodes void*
	27	swap *// retrait de y*
	28	pop
}	29	swap *// retrait de f*
	30	pop
	31	swap *// retrait de y*
	32	pop
	33	pop *// retrait du main*
	34	jcstop

Contrôleur de type

Le système de typage : Statique

- *Contrôle de type :* incompatibilité de type des opérandes.
- *Contrôle du flot d'exécution :* transfert du contrôle en un autre point du programme.
- *Contrôle d'unicité ou de répétition :* un nom peut devoir apparaître une nombre fixé de fois.



Réalisé : un passe ou plusieurs si surcharges ou polymorphismes.

Contrôle de Type

Contrôle de type = **Spécification du langage**

Expressions de type :

- *Type de base :* Booléen, Caractère, entier, réel, void...
- *Noms de type*
- *Constructeurs de type :*
  - Tableau
  - Produit
  - Enregistrement
  - Pointeur
  - Fonction...
- Variables pour définir des *expressions de type*.

Les systèmes de typages

- Mis en œuvre par le contrôle de type
- Choix par le concepteur du langage (Ada, Pascal, C...)
- Éléments externes pour le débuggage : lint
- Équivalence de type : Structurelle, nominale

Contrôle statique vs dynamique :

- Langage
- Éléments seulement connus pendant l'exécution

Récupération des erreurs :

- Technique de récupération identique à l'analyse syntaxique.

Typage MiniJaja / JajaCode

MiniJaja :

- Table des symboles ou dictionnaire de données
- Contrôle de type statique :
  - Pré-déclaration : Variables, méthodes
  - Paramètre : nombre et type
  - Opérateur spécifique : booléen et entier
  - Valeur des expressions : `return`, condition
- Contrôle de type dynamique : indice

Règles :

- $\text{MEM} \vdash \text{INSTR} \rightarrow \text{MEM}$
- $\text{MEM} \overset{eval}{\vdash} \text{EXP} \Rightarrow \text{SORTE}$

JajaCode :

- Contrôle statique : adresse, variable/opération
- Contrôle dynamique : pile

Preuve de correction

Un **compilateur correct** est un compilateur qui transforme un programme MiniJaja en un programme Jaja-Code tels que leurs sémantiques interprétatives soient "équivalentes".

Signification de sémantique interprétative "équivalentes" :

1. Identité des calculs (succession d'états mémoire)
2. Identité des derniers états mémoires
3. Identité des valeurs associées au résultat du programme dans les états mémoire des deux calculs.

Équivalence

Équivalence – noté équiv(m, m', i)

Soient m, $m' \in MEM$ deux états mémoire et i un identificateur. L'état mémoire m est équivalent à l'état mémoire m' pour l'objet i si et seulement si :

$$Val(i, m) = Val(i, m')$$

Preuve

La preuve de correction d'un compilateur c d'un langage L dans un langage L' (dont le domaine de leurs sémantique interprétative est l'ensemble des séquences d'états mémoire appartenant MEM) consiste à prouver que $\forall PN \in L, \forall m \in MEM, \forall i \in ID$:

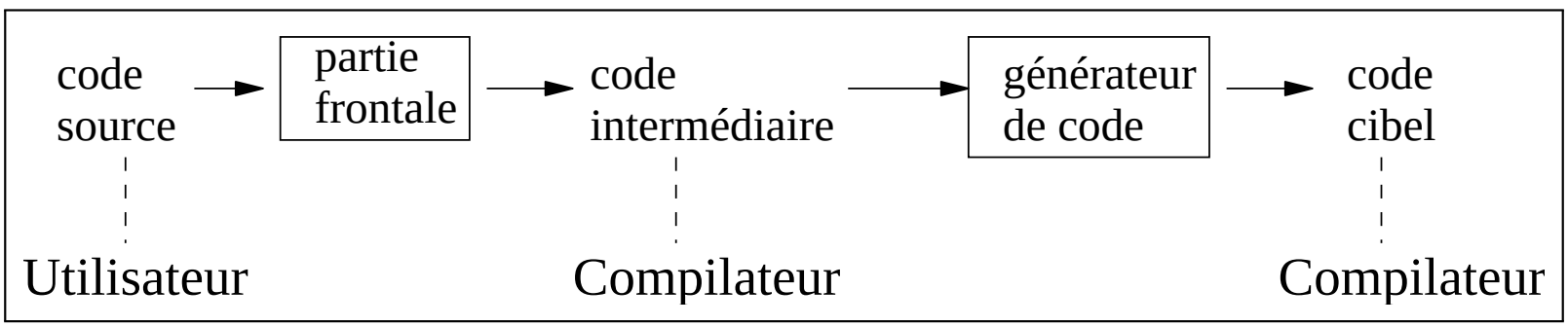
- $S_{opL}(PN, m) = m' \ \wedge \ S_{opL'}(c[PN], m) = m_0, m_1, \ldots m'' \ \Rightarrow$ équiv(m', m'', i) C'est à dire :
- $m \vdash PN \Rightarrow m'$,
- $a \vdash PN \Rightarrow \{PC, n\}$
- $< m, a > \vdash PC \twoheadrightarrow < m'', a' >$

Optimisation de code

Critères :

- Préservation de la signification du programme
- Accélérer les programmes
- Rapport coût / gain intéressant

Il existe trois niveaux d'optimisations :



Optimisation

Utilisateur :

- Changer d'algorithme
- Transformer le code
- Les boucles

Compilateur :

Code intermédiaire :	*Code Cible :*
▶ Boucles	▶ Utiliser les registres
▶ Appels de procédures	▶ Sélectionner instructions
▶ Calculs des adresses	▶ Transformations locales

Graphe de flots de contrôle

Partition en bloc de base

Entrée : Une séquence d'instructions Jaja-Code
Sortie : Une liste de bloc de base telle que chaque instruction du Jaja-Code appartient exactement à un bloc.

1. Déterminer les instructions de tête de bloc :
   - 1.1 La première instruction
   - 1.2 Toute instruction atteint par branchement
   - 1.3 Toute instruction qui suit un branchement
2. Un bloc correspond aux instructions commençant par un bloc de tête jusquèau bloc de tête suivant (exclus).

Ordonnancement des nœuds

Tant qu'il reste des nœuds internes non traités **faire**
Début
    Choisir une nœud n non énuméré
        dont les parents ont été énumérés
    Afficher n
    Tant que le fils m de n la plus à gauche $\neq$ une feuille
        et a ses parents déjà énumérés **faire**
    Début
        Afficher m
        $n \leftarrow m$
    Fin
Fin

Nombre de Registres

Début
  Si n est une feuille **Alors**
    Si n est le fils gauche de son père **Alors**
      Étiquette(n) $\leftarrow$ 1
    Sinon
      Étiquette(n) $\leftarrow$ 0
  Sinon Début
    Soient $n_1, n_2 \ldots, n_k$ les fils de n ordonnés par Étiquette décroissante
    Étiquette(n) $\leftarrow \max_{1 \leq i \leq k}$ (Étiquette(n_i)+i-1)
  Fin
Fin