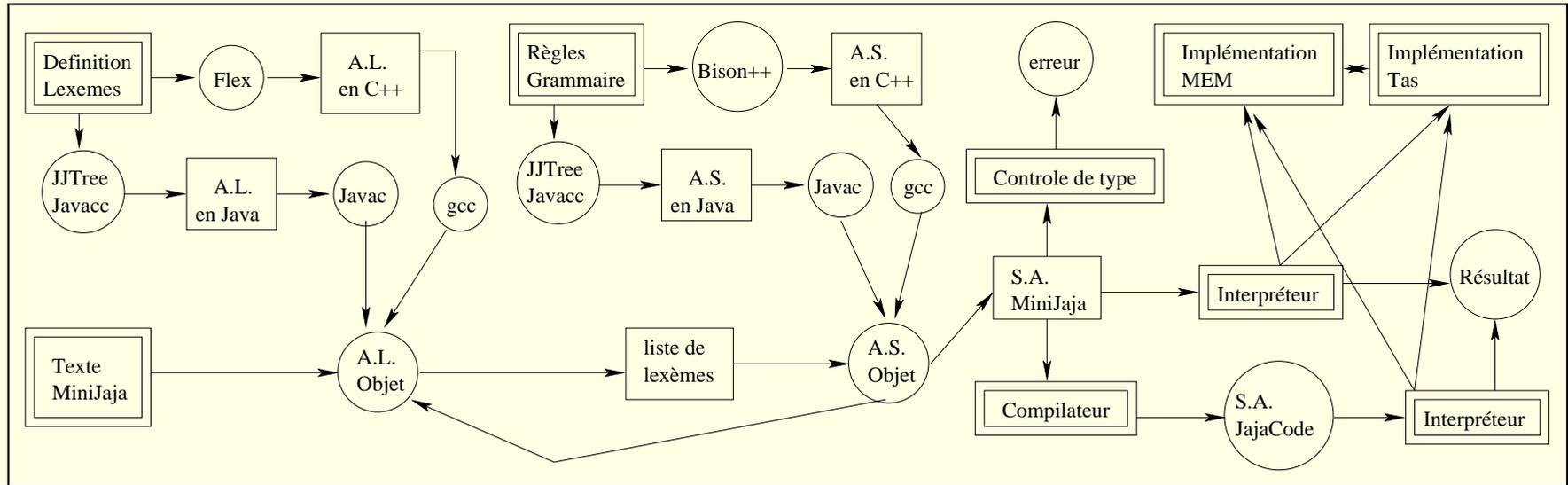




L I F C

Compilation - Interprétation

UNIVERSITÉ DE FRANCHE-COMTÉ



F. Bouquet

Master S&T - Mention Informatique
1^{ère} année

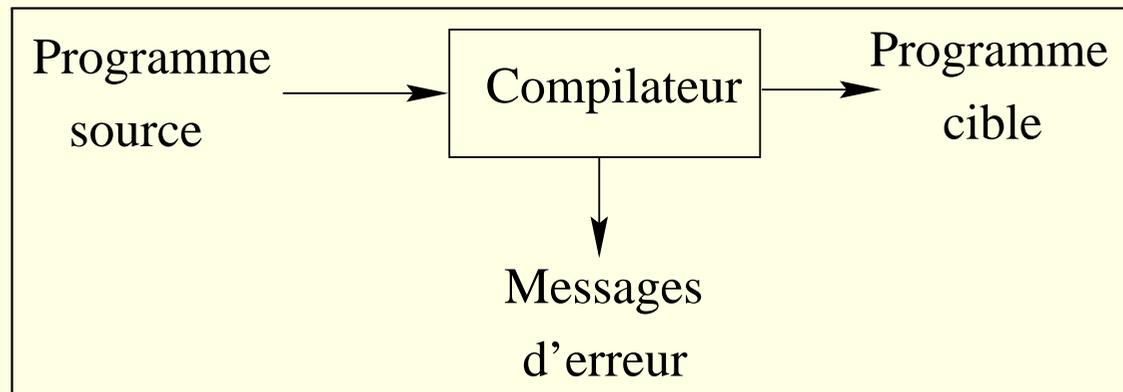
<http://lifc.univ-fcomte.fr/~bouquet/Enseignement>



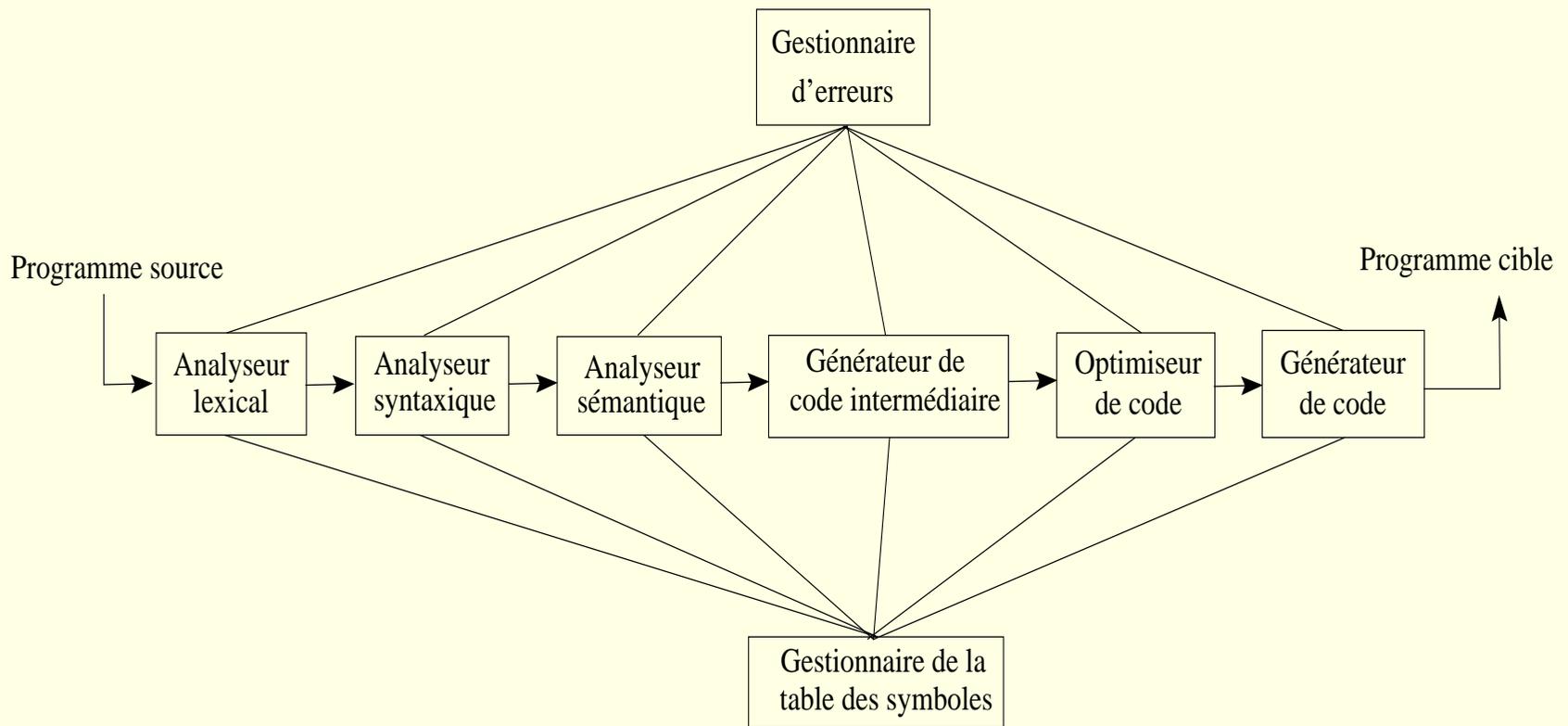
FRE 2661

Traducteur (Source / Destination) :

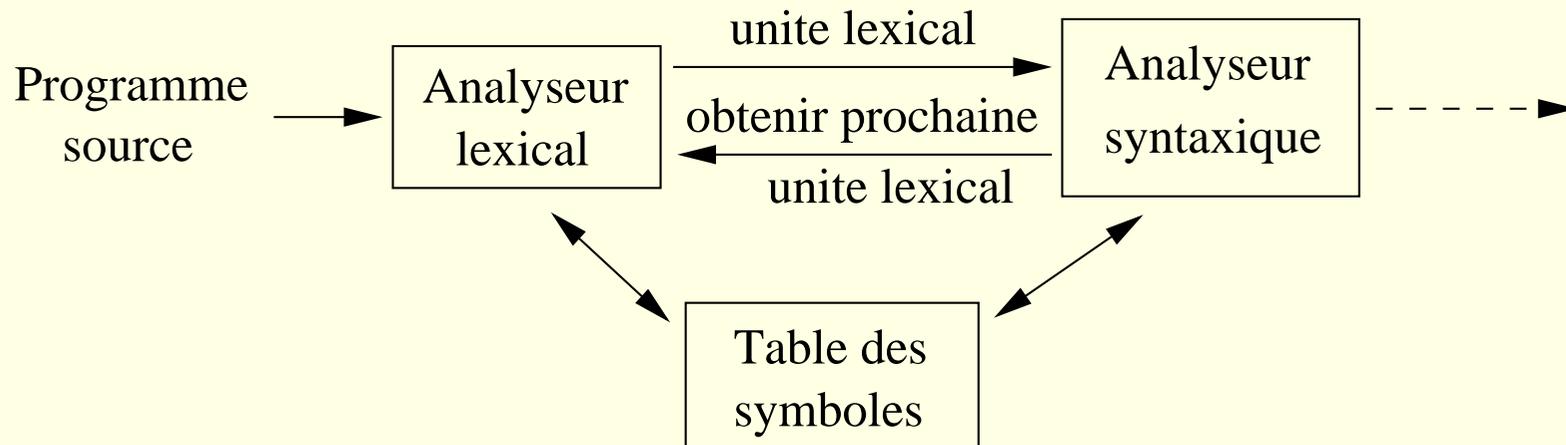
- ① Langage de hauts niveaux → Assembleur
- ② Système de composition de textes
- ③ Compilateur de silicium
- ④ Interprètes de requêtes



Phase de la compilation



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)

🎯 **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

- ① 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
- ① 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^+
- ① Expressions régulières

🌀 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 **Backus-Naur Form** (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, $G = \langle X, V, P, S \rangle$:

- LL(k) : Left most derivation pour la construction de l'arbre
- LR(k) : Right most derivation pour la construction de l'arbre
 - Left to right pour l'analyse du texte
 - k symboles de la chaîne d'entrée

Analyse descendante LL(1)

⊙ Propriétés :

- Non réursive à gauche
- Non ambiguë

⊙ 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.

Syntaxe abstraite MiniJaja

classe :	ID × DECLS × MAIN	→ CLASSE
ident :	<u>string</u>	→ 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 :	$\text{INSTR} \times \text{INSTRS}$	$\rightarrow \text{INSTRS}$
inil :		$\rightarrow \text{INSTRS}$
affectation :	$\text{ID} \times \text{EXP}$	$\rightarrow \text{INSTR}$
somme, incrément :	ID	$\rightarrow \text{INSTR}$
appel :	$\text{ID} \times \text{LISTEXP}$	$\rightarrow \text{INSTR}$
retour :	EXP	$\rightarrow \text{INSTR}$
si :	$\text{EXP} \times \text{INSTRS} \times \text{INSTRS}$	$\rightarrow \text{INSTR}$
tantque :	$\text{EXP} \times \text{INSTRS}$	$\rightarrow \text{INSTR}$
listexp :	$\text{EXP} \times \text{LISTEXP}$	$\rightarrow \text{LISTEXP}$
exnil :		$\rightarrow \text{LISTEXP}$
appelE :	$\text{ID} \times \text{LISTEXP}$	$\rightarrow \text{EXP}'$
tab :	$\text{ID} \times \text{EXP}$	$\rightarrow \text{ID}$
non, moins :	EXP	$\rightarrow \text{EXP}'$
ou, et, =, > :	$\text{EXP} \times \text{EXP}$	$\rightarrow \text{EXP}'$
*, /, +, - :	$\text{EXP} \times \text{EXP}$	$\rightarrow \text{EXP}'$
nbre :	<u>integer</u>	$\rightarrow \text{EXP}'$
vrai, faux :		$\rightarrow \text{EXP}'$

La **syntaxe abstraite** consiste à définir :

- ① un ensemble de noms de fonction dont on donne le profil
- ② un ensemble de types de fonctions permettant de typer les fonctions.

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

- ① à chaque règle de la syntaxe concrète est associé un opérateur
- ② à chaque non terminal de la syntaxe concrète est associé un type d'opérateur



Table des symboles

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

Une **entrée** doit être composée :

- ① Nom
- ① Attributs

Action sur les éléments :

- ① Rechercher,
- ① Ajouter,
- ① Supprimer

Structure de données en liste

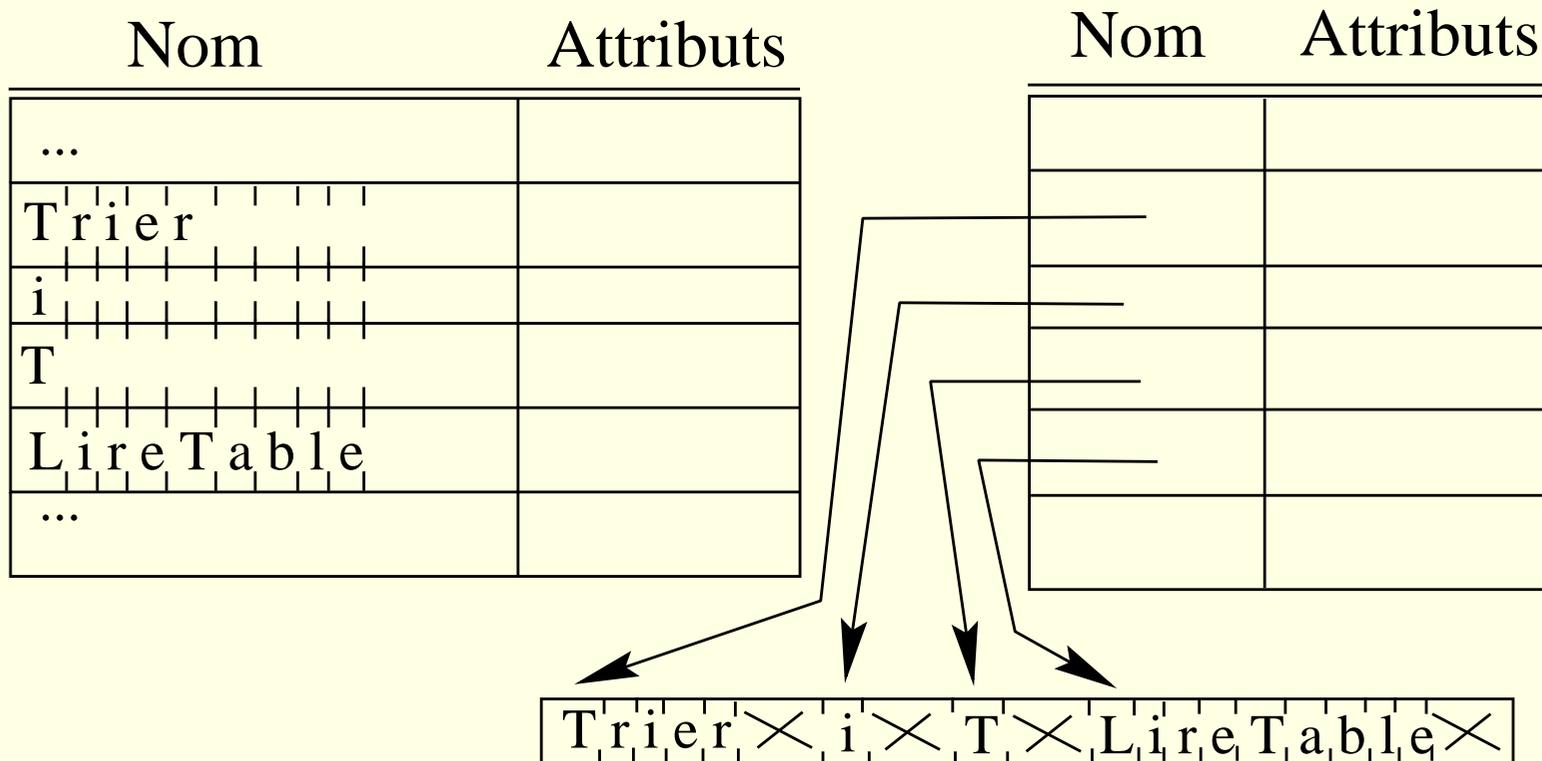
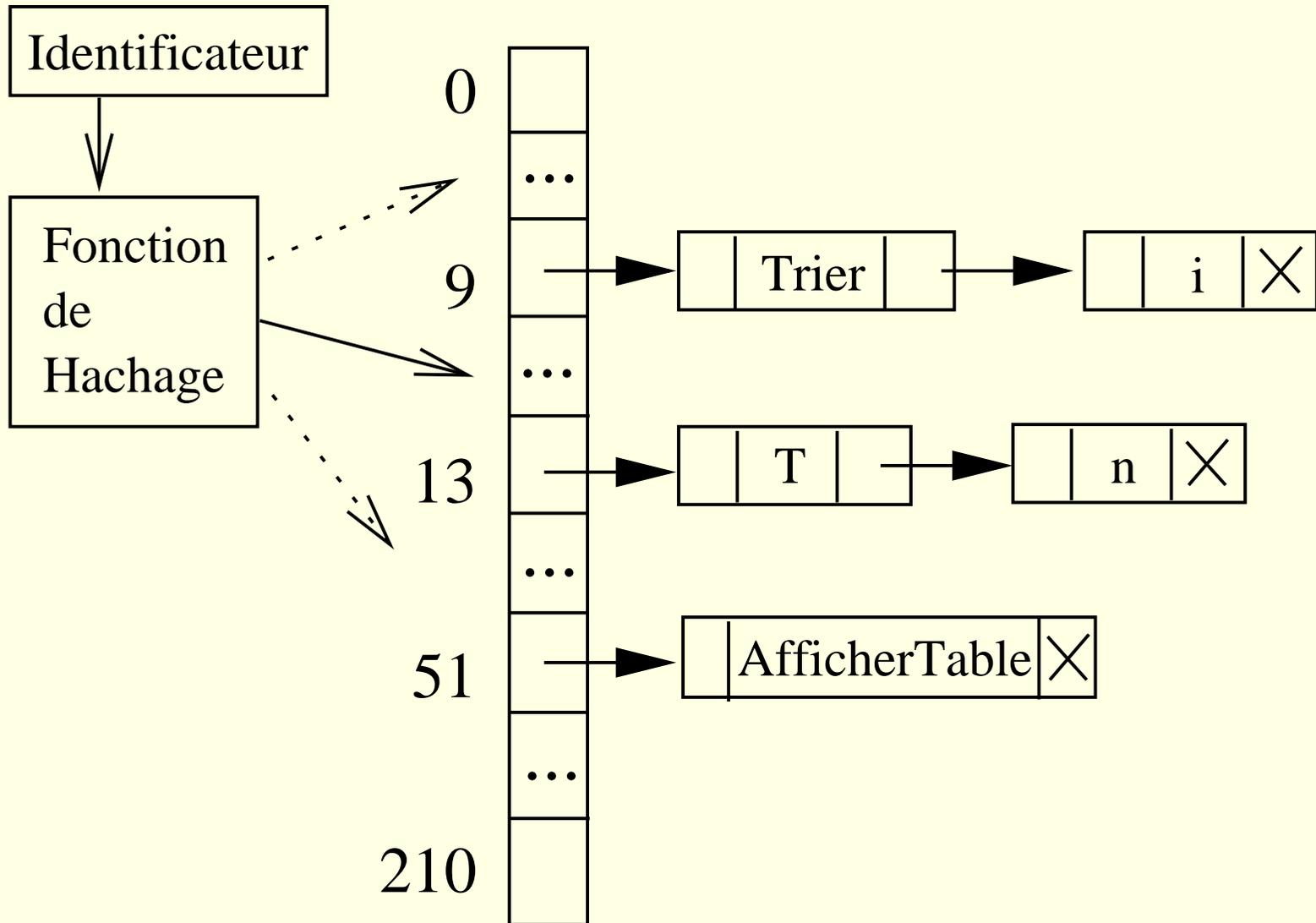


Table de Hachage



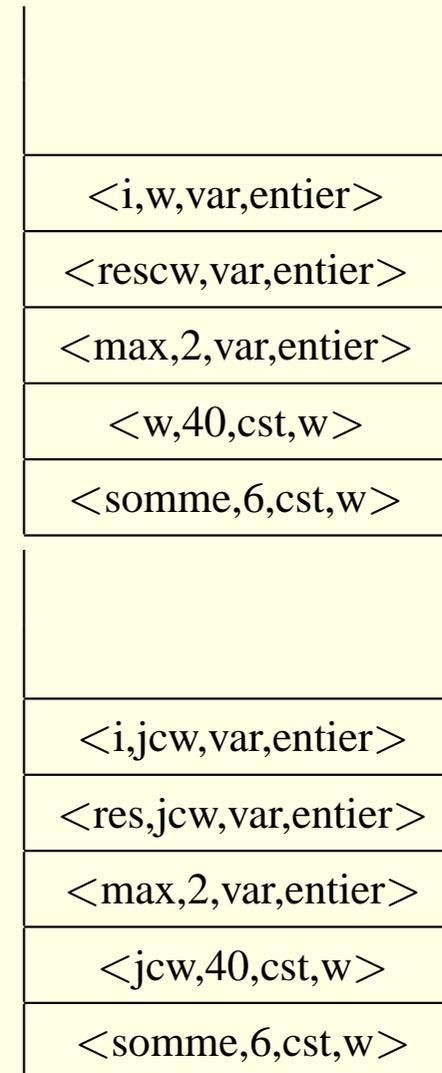
Pile d'exécution

Rôle :

- ① Mémoire pour les variables
- ① Sauvegarde de l'environnement
- ① Calcul temporaire

Utilisation :

- ① Interpréteur(s)
- ① Calculs dynamiques
- ① Lien avec la table des symboles





Ramasse miettes

But : Gérer la mémoire d'une façon efficace

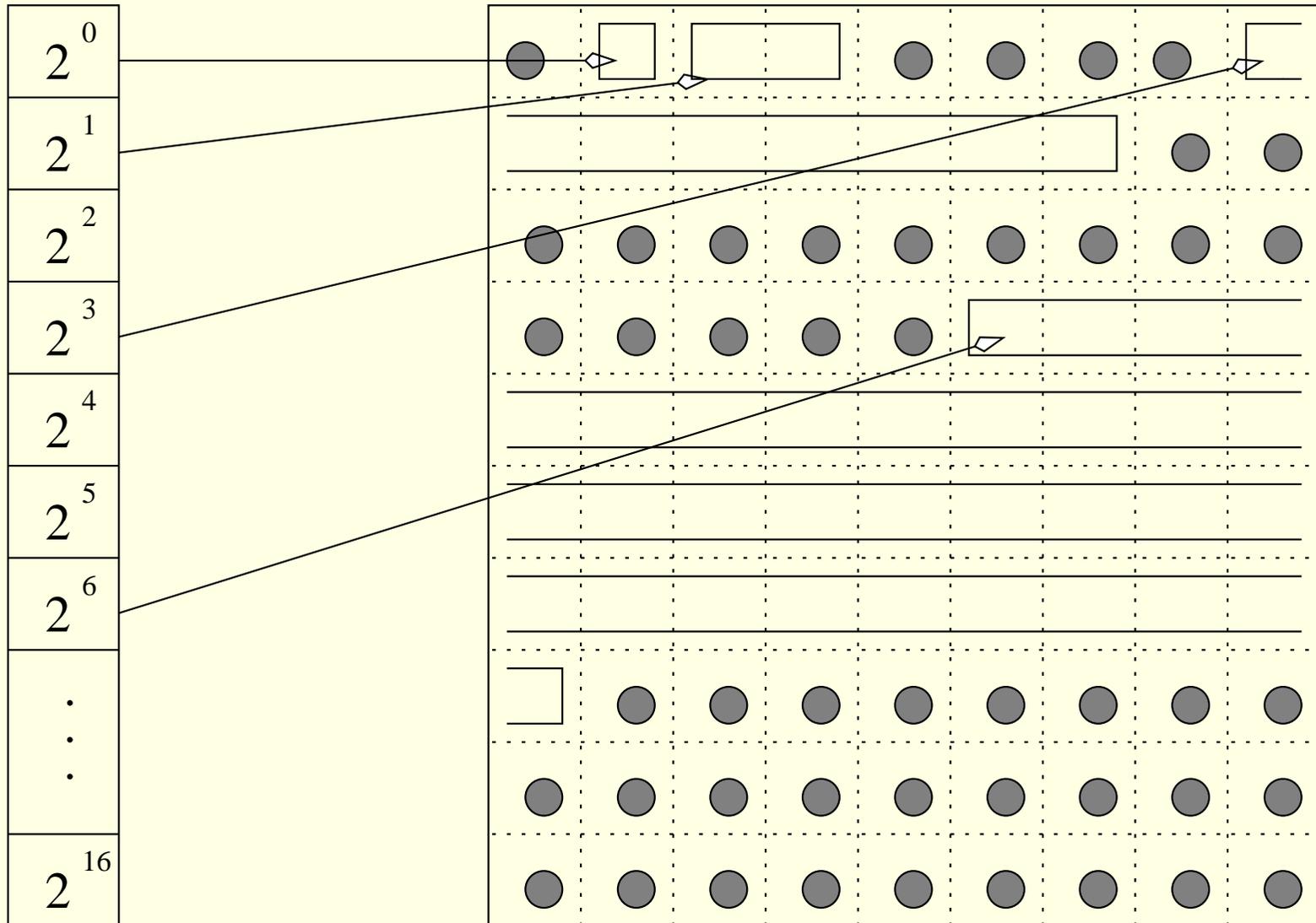
Entrée :

- ① un type de mémoire
- ① les objets manipulés

Action sur les éléments :

- ① Rechercher,
- ① Ajouter,
- ① Supprimer

Exemple de gestion tas



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

Propriétés :

- ③ Mi-chemin entre la pile et la table
- ③ Pile pour le masquage
- ③ Table pour l'accès direct aux attributs

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

- ③ ID : l'ensemble des identificateurs
- ③ VAL : l'ensemble des valeurs associées
- ③ OBJ : la nature de l'objet représenté
- ③ $SORTE$: le type de l'objet

Prototypage (1)

Gestion de la pile :

Créer, [] :		→ MEM
Empiler, . :	QUA × MEM	→ MEM
Dépiler :	MEM	→ MEM
Echanger :	MEM	→ MEM

Opérations à partir des constructeurs :

DeclVar :	ID × VAL × SORTÉ × MEM	→ MEM
IdentVal :	ID × SORTÉ × MEM × NAT	→ MEM
DeclCst :	ID × VAL × SORTÉ × MEM	→ MEM
DeclTab :	ID × VAL × SORTÉ × MEM	→ MEM
DeclMeth :	ID × VAL × SORTÉ × MEM	→ MEM

Axiome de Gestion de la pile :

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

Opérations de modification de la mémoire :

AffecterVal : $ID \times VAL \times MEM \rightarrow MEM$

AffecterValT : $ID \times VAL \times VAL \times MEM \rightarrow MEM$

AffecterType : $ID \times SORT E \times MEM \rightarrow MEM$

RetirerDecl : $ID \times MEM \rightarrow MEM$

ExpParam : $VAL \times VAL \times MEM \rightarrow MEM$

Déclaration :

⊙ $RetirerDecl(i, []) = []$

⊙ $RetirerDecl(i, \langle i_1, v_1, o, t \rangle .m) = \mathbf{Si} \ i == i_1 \ \mathbf{alors}$
 $(\mathbf{Si} \ o == tab \ \mathbf{alors} \ RetirerTas(v_1, t) \ \mathbf{endif} \ m) \ \mathbf{sinon}$
 $\langle i_1, v_1, o, t \rangle .RetirerDecl(i, m)$

Affectation :

- ① $AffecterVal(i, v, \langle i_1, v_1, o, t \rangle .m) = \mathbf{Si} \ i \neq i_1 \ \mathbf{alors}$
 $\langle i_1, v_1, o, t \rangle .AffecterVal(i, v, m) \ \mathbf{sinon} \ \mathbf{Si} \ o == vcst \ \mathbf{alors}$
 $\langle i, v, cst, t \rangle .m \ \mathbf{sinon} \ \mathbf{Si} \ o == cst \ \mathbf{alors} \ m \ \mathbf{sinon} \ \langle i, v, o, t \rangle .m$
- ① $AffecterValT(i, v, v_0, \langle i_1, v_1, o, t \rangle .m) = \mathbf{Si} \ i \neq i_1 \ \mathbf{alors}$
 $\langle i_1, v_1, o, t \rangle .AffecterValT(i, v, v_0, m) \ \mathbf{sinon} \ AffecterTas(v_1, v, v_0)$
- ① $AffecterType(i, t, \langle i_1, v_1, o, t_1 \rangle .m) = \mathbf{Si} \ i == i_1 \ \mathbf{alors}$
 $\langle i, v_1, o, t \rangle .m \ \mathbf{sinon} \ \langle i_1, v_1, o, t_1 \rangle .AffecterType(i, t, m)$

Paramètre :

- ① $ExpParam(lexp, ent, m) = \mathbf{Si} \ (lexp \neq exnil) \ \mathbf{et} \ (ent \neq enil) \ \mathbf{alors}$
 $ExpParam(le_1, ents, DeclVar(i, v, t, m)) \ \mathbf{sinon} \ m$
/ où v est définie par $m \stackrel{eval}{\vdash} e \Rightarrow v$, on suppose que : $lexp = listexp(e, le_1)$ et $ent = entetes(entete(t, i), ents)$ */*

Opérations d'accès aux informations de la mémoire :

$$\text{Val} : \text{ID} \times \text{MEM} \rightarrow \text{VAL} \cup \{w\}$$

$$\text{ValT} : \text{ID} \times \text{VAL} \times \text{MEM} \rightarrow \text{VAL} \cup \{w\}$$

$$\text{Objet} : \text{ID} \times \text{MEM} \rightarrow \text{OBJ} \cup \{w\}$$

$$\text{Sorte} : \text{ID} \times \text{MEM} \rightarrow \text{SORTE} \cup \{w\}$$

$$\text{Paramètre} : \text{ID} \times \text{MEM} \rightarrow \text{DECLS} \cup \{w\}$$

$$\text{Déclaration} : \text{ID} \times \text{MEM} \rightarrow \text{DECLS} \cup \{w\}$$

$$\text{Corps} : \text{ID} \times \text{MEM} \rightarrow \text{INSTRS} \cup \{w\}$$

🌀 $\text{Val}(i, []) = w$

$$\text{Val}(i, \langle i_1, v_1, o, t \rangle .m) = \mathbf{Si} \ i == i_1 \ \mathbf{alors} \ v_1 \ \mathbf{sinon} \ \text{Val}(i, m)$$

🌀 $\text{ValT}(i, v, []) = w$

$$\text{ValT}(i, v, \langle i_1, v_1, o, t \rangle .m) = \mathbf{Si} \ i == i_1 \ \mathbf{alors} \ \text{ValeurTas}(v_1, v) \\ \mathbf{sinon} \ \text{ValT}(i, v, m)$$



Axiome (3)

- ① $Objet(i, []) = w$
 $Objet(i, \langle i_1, v_1, o, t \rangle .m) = \mathbf{Si} \ i == i_1 \ \mathbf{alors} \ o \ \mathbf{sinon} \ Objet(i, m)$
- ① $Sorte(i, []) = w$
 $Sorte(i, \langle i_1, v_1, o, t \rangle .m) = \mathbf{Si} \ i == i_1 \ \mathbf{alors} \ t \ \mathbf{sinon} \ Sorte(i, m)$
- ① $Param\grave{e}tre(i, []) = w$
 $Param\grave{e}tre(i, \langle i_1, v_1, o, t \rangle .m) = \mathbf{Si} \ i \neq i_1 \ \mathbf{alors} \ Param\grave{e}tre(i, m) \ \mathbf{sinon}$
 $\mathbf{Si} \ o \neq meth \ \mathbf{alors} \ w \ \mathbf{sinon} \ \mathbf{Si} \ v_1 == m\acute{e}thode(t, i, ents, dv, is) \ \mathbf{alors}$
 $ents \ \mathbf{sinon} \ w$
- ① $D\acute{e}claration(i, []) = w$
 $D\acute{e}claration(i, \langle i_1, v_1, o, t \rangle .m) = \mathbf{Si} \ i \neq i_1 \ \mathbf{alors} \ D\acute{e}claration(i, m)$
 $\mathbf{sinon} \ \mathbf{Si} \ o \neq meth \ \mathbf{alors} \ w \ \mathbf{sinon} \ \mathbf{Si} \ v_1 == m\acute{e}thode(t, i, ents, dv, is)$
 $\mathbf{alors} \ dv \ \mathbf{sinon} \ w$
- ① $Corps(i, []) = w$
 $Corps(i, \langle i_1, v_1, o, t \rangle .m) = \mathbf{Si} \ i \neq i_1 \ \mathbf{alors} \ Corps(i, m) \ \mathbf{sinon}$
 $\mathbf{Si} \ o \neq proc \ \mathbf{alors} \ w \ \mathbf{sinon} \ \mathbf{Si} \ v_1 == m\acute{e}thode(t, i, ents, dv, is) \ \mathbf{alors} \ is$
 $\mathbf{sinon} \ w$

Sémantique interprétative

MEM \vdash INSTR \rightarrow MEM



$$\text{DeclVar}(i, w, w, []) \vdash dss \rightarrow m_1$$

$$m_1 \vdash mma \rightarrow m_2$$

retrait

$$m_2 \vdash dss \rightarrow m_3$$

$$[\text{classe}] : \frac{\text{RetirerDecl}(\text{ident}(i), m_3) \rightarrow m_4}{[] \vdash \text{classe}(\text{ident}(i), dss, mma) \rightarrow m_4}$$



$$[\text{tableau}] : m \vdash \text{tableau}(t, \text{ident}(i), v) \rightarrow \text{DeclTab}(i, v, t, m)$$



$$[\text{decls}] : \frac{m \vdash ds \rightarrow m_1, m_1 \vdash dss \rightarrow m_2}{m \vdash \text{decls}(ds, dss) \rightarrow m_2}$$



$$[\text{vars}] : \frac{m \vdash dv \rightarrow m_1, m_1 \vdash dvs \rightarrow m_2}{m \vdash \text{vars}(dv, dvs) \rightarrow m_2}$$

S. Interprétative (Suite...)



$$m \vdash dvs \rightarrow m_1$$

$$m_1 \vdash iss \rightarrow m_2$$

$$[\text{main}] : \frac{\text{retrait} \quad m_2 \vdash dvs \rightarrow m_3}{m \vdash \text{main}(dvs, iss) \rightarrow m_3}$$



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



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



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



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

S. Interprétative (...Suite...)

$$\textcircled{a} \text{ [instrs]} : \frac{m \vdash is \rightarrow m_1, m_1 \vdash iss \rightarrow m_2}{m \vdash \text{instrs}(is, iss) \rightarrow m_2}$$

$$\textcircled{a} \text{ [inil]} : m \vdash \text{inil} \rightarrow m$$

$$\textcircled{a} \text{ [affectation]} : \frac{m \stackrel{eval}{\vdash} e \Rightarrow v}{m \vdash \text{affectation}(\text{ident}(i), e) \rightarrow \text{AffecterVal}(i, v, m)}$$

$$\textcircled{a} \text{ [affectationT]} : \frac{m \stackrel{eval}{\vdash} e \Rightarrow v, m \stackrel{eval}{\vdash} e_1 \Rightarrow ind}{m \vdash \text{affectation}(\text{tab}(\text{ident}(i), e_1), e) \rightarrow \text{AffecterValT}(i, ind, v, m)}$$

$$\textcircled{a} \text{ [incrément]} :$$

$$m \vdash \text{incrément}(\text{ident}(i)) \rightarrow \text{AffecterVal}(i, \text{Val}(i, m) + 1, m)$$

$$\textcircled{a} \text{ [incrémentT]} :$$

$$\frac{m \stackrel{eval}{\vdash} e \Rightarrow ind}{m \vdash \text{incrément}(\text{tab}(\text{ident}((i), e)) \rightarrow \text{AffecterValT}(i, ind, \text{ValT}(i, ind, m) + 1, m)}$$

S. Interprétative (...Suite...)



$\text{ExpParam}(lexp, \text{Paramètre}(i, m), m) \vdash \text{Déclaration}(i, m) \rightarrow m_1$

$m_1 \vdash \text{Corps}(i, m) \rightarrow m_2$

retrait

$m_2 \vdash \text{Déclaration}(i, m) \rightarrow m_3$

retrait

[appel] : $\frac{m_3 \vdash \text{Paramètre}(i, m) \rightarrow m_4}{m \vdash \text{appel}(\text{ident}(i), lexp) \rightarrow m_4}$



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



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



[sommeT] :

$\frac{m \vdash e \Rightarrow v, m \vdash e_1 \Rightarrow ind}{m \vdash \text{somme}(\text{tab}(\text{ident}(i), e_1), e) \rightarrow \text{AffecterValT}(i, ind, \text{ValT}(i, ind, m) + v, m)}$

S. Interprétative (...Suite)

$$\textcircled{1} \quad [\text{sivrai}] : \frac{m \stackrel{eval}{\vdash} e \Rightarrow true, m \vdash iss \rightarrow m_1}{m \vdash \text{si}(e, iss, iss_1) \rightarrow m_1}$$

$$\textcircled{2} \quad [\text{sifaux}] : \frac{m \stackrel{eval}{\vdash} e \Rightarrow false, m \vdash iss_1 \rightarrow m_1}{m \vdash \text{si}(e, iss, iss_1) \rightarrow m_1}$$

$$\textcircled{3} \quad [\text{tantquevrai}] : \frac{m \stackrel{eval}{\vdash} e \Rightarrow true, m \vdash iss \rightarrow m_1, m_1 \vdash \text{tantque}(e, iss) \rightarrow m_2}{m \vdash \text{tantque}(e, iss) \rightarrow m_2}$$

$$\textcircled{4} \quad [\text{tantquefaux}] : \frac{m \stackrel{eval}{\vdash} e \Rightarrow false}{m \vdash \text{tantque}(e, iss) \rightarrow m}$$



eval

MEM \vdash EXP \rightarrow VAL

$$\textcircled{a} \quad [\text{tab}] \frac{\text{eval} \quad m \vdash e \Rightarrow v}{\text{eval} \quad m \vdash \text{tab}(\text{ident}(i), e) \Rightarrow \text{ValT}(i, v, m)}$$

$$\textcircled{a} \quad [\text{vrai}] : m \text{eval} \vdash \text{vrai} \Rightarrow \text{true}$$

$$\textcircled{a} \quad [\text{non}] : \frac{\text{eval} \quad m \vdash e \Rightarrow v}{\text{eval} \quad m \vdash \text{non}(e) \Rightarrow \neg v}$$

$$\textcircled{a} \quad [\text{faux}] : m \text{eval} \vdash \text{faux} \Rightarrow \text{false}$$

$$\textcircled{a} \quad [\text{moins}] : \frac{\text{eval} \quad m \vdash e \Rightarrow v}{\text{eval} \quad m \vdash \text{moins}(e) \Rightarrow -v}$$

$$\textcircled{a} \quad [\text{nbre}] : m \text{eval} \vdash \text{nbre}(n) \Rightarrow n$$

$$\textcircled{a} \quad [\text{op2}] : \frac{\text{eval} \quad m \vdash e \Rightarrow v, \text{eval} \quad m \vdash e_1 \Rightarrow v_1}{\text{eval} \quad m \vdash \text{op2}(e, e_1) \Rightarrow v \text{ op } v_1}$$

$$\textcircled{a} \quad [\text{omega}] : m \text{eval} \vdash \text{omega} \Rightarrow w$$

$$\textcircled{a} \quad [\text{ident}] : m \text{eval} \vdash \text{ident}(i) \Rightarrow \text{Val}(i, m)$$

$$\textcircled{a} \quad [\text{appelE}] : \frac{m \vdash \text{appel}(\text{ident}(i), \text{lexp}) \rightarrow m_1, m_1 \text{eval} \vdash \text{VariableClasse}(m_1) \Rightarrow v}{\text{eval} \quad m \vdash \text{appelE}(\text{ident}(i), \text{lexp}) \Rightarrow v}$$

retrait
MEM ⊢ DECLS ∪ ENTETES ∪ ENTETE → MEM

⊙ [rdecls] :
$$\frac{\text{retrait} \quad m \vdash dss \rightarrow m_1, \quad \text{retrait} \quad m_1 \vdash ds \rightarrow m_2}{\text{retrait} \quad m \vdash \text{decls}(ds, dss) \rightarrow m_2}$$

⊙ [renêtes],[rvars] :
$$\frac{\text{retrait} \quad m \vdash dvs \rightarrow m_1, \quad \text{retrait} \quad m_1 \vdash dv \rightarrow m_2}{\text{retrait} \quad m \vdash \text{vars}(dv, dvs) \rightarrow m_2}$$

⊙ [renil],[rvnil],[rmnil] :
$$\text{retrait} \quad m \vdash \text{mnil} \rightarrow m$$

⊙ [rcst],[rentête],[rvar] :
$$\text{retrait} \quad m \vdash \text{var}(t, \text{ident}(i), e) \rightarrow \text{RetirerDecl}(i, m)$$

⊙ [rtableau] :
$$\text{retrait} \quad m \vdash \text{tableau}(t, \text{ident}(i), e) \rightarrow \text{RetirerDecl}(i, m)$$

⊙ [rméthode] :
$$\text{retrait} \quad m \vdash \text{méthode}(t, \text{id}(i), en, dvs, ins) \rightarrow \text{RetirerDecl}(i, m)$$

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	<i>JajaCode(\$2,\$4)</i>
	<u>vide</u>	<i>jcnil</i>
instrs ::=	<u>init</u> <u>swap</u>	<i>init</i> <i>swap</i>
	<u>new</u> (ident, type, sorte, adr)	<i>new(\$3,\$5,\$7,\$9)</i>
	<u>newarray</u> (ident, type)	<i>newarray(\$3,\$5)</i>
	<u>invoke</u> (ident)	<i>invoke(\$3)</i>
	<u>return</u>	<i>return</i>
	<u>push</u> (valeur) <u>pop</u>	<i>push(\$3)</i> <i>pop</i>
	<u>load</u> (ident) <u>aload</u> (ident)	<i>load(\$3)</i> <i>aload(\$3)</i>
	<u>store</u> (ident) <u>astore</u> (ident)	<i>store(\$3)</i> <i>astore(\$3)</i>
	<u>if</u> (adresse) <u>goto</u> (adresse)	<i>if(\$3)</i> <i>goto(\$3)</i>

Syntaxe concrète (Suite)

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

Syntaxe abstraite du Jaja-Code

JajaCode :	JCODE \times JCODES	\rightarrow JCODES
jcnil :		\rightarrow JCODES
init, pop, nop, jcstop, return, swap, oper :		\rightarrow JCODE
invoke, inc, ainc :	ID	\rightarrow JCODE
load, store, aload, astore :	ID	\rightarrow JCODE
new :	ID \times TYPE \times SORTÉ \times JCVAL	\rightarrow JCODE
newarray :	ID \times TYPE	\rightarrow JCODE
push :	JCVAL	\rightarrow JCODE
goto, if :	ADR	\rightarrow JCODE
jcident :	<u>string</u>	\rightarrow ID
jcnbre :	<u>entier</u>	\rightarrow JCVAL
jcvtrai, jcfaux, jcw :		\rightarrow JCVAL

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

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



Sémantique des instructions

$\langle MEM, ADR \rangle \vdash JCODE \twoheadrightarrow \langle MEM, ADR \rangle$

- [init] : $\langle m, a \rangle \vdash \text{init} \twoheadrightarrow \langle [], a+1 \rangle$
- [jcstop] : $\langle m, a \rangle \vdash \text{jcstop} \twoheadrightarrow \langle m, \perp \rangle$
- [nop] : $\langle m, a \rangle \vdash \text{nop} \twoheadrightarrow \langle m, a+1 \rangle$
- [swap] : $\langle q_1.q_2.m, a \rangle \vdash \text{swap} \twoheadrightarrow \langle q_2.q_1.m, a+1 \rangle$
- [newV] : $\langle m, a \rangle \vdash \text{new}(i, t, \text{var}, s) \twoheadrightarrow \langle \text{IdentVal}(i, t, m, s), a+1 \rangle$
- [newC] : $\langle \langle w, v, \text{cst}, w \rangle . m, a \rangle \vdash \text{new}(i, t, \text{cst}, s) \twoheadrightarrow \langle \text{DeclCst}(i, v, t, m), a+1 \rangle$
- [newM] : $\langle \langle w, v, \text{cst}, w \rangle . m, a \rangle \vdash \text{new}(i, t, \text{meth}, s) \twoheadrightarrow \langle \text{DeclMeth}(i, v, t, m), a+1 \rangle$
- [newarray] : $\langle \langle w, v, \text{cst}, w \rangle . m, a \rangle \vdash \text{newarray}(i, t) \twoheadrightarrow$
 $\langle \text{DeclTab}(i, v, t, m), a+1 \rangle$

S. instructions Jaja-Code

- [invoke] : $\langle m, a \rangle \vdash \text{invoke}(i) \rightarrow \langle \langle w, a+1, \text{cst}, w \rangle . m, \text{Val}(i, m) \rangle$
- [return] : $\langle \langle w, a_1, \text{cst}, w \rangle . m, a \rangle \vdash \text{return} \rightarrow \langle m, a_1 \rangle$
- [goto] : $\langle m, a \rangle \vdash \text{goto}(a_1) \rightarrow \langle m, a_1 \rangle$
- [load] : $\langle m, a \rangle \vdash \text{load}(i) \rightarrow \langle \langle w, \text{Val}(i, m), \text{cst}, w \rangle . m, a+1 \rangle$
- [aload] : $\langle \langle w, \text{ind}, \text{cst}, w \rangle . m, a \rangle \vdash \text{aload}(i) \rightarrow$
 $\langle \langle w, \text{ValT}(i, \text{ind}, m), \text{cst}, w \rangle . m, a+1 \rangle$
- [store] : $\langle \langle w, v, \text{cst}, w \rangle . m, a \rangle \vdash \text{store}(i) \rightarrow \langle \text{AffecterVal}(i, v, m), a+1 \rangle$
- [astore] : $\langle \langle w, v, \text{cst}, w \rangle . \langle w, \text{ind}, \text{cst}, w \rangle . m, a \rangle \vdash \text{astore}(i) \rightarrow$
 $\langle \text{AffecterValT}(i, \text{ind}, v, m), a+1 \rangle$

S. instructions Jaja-Code

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

Comp. MiniJaja \rightarrow Jaja-Code

Compilateur :

- ① A tout programme MiniJaja associe un programme Jaja-Code
- ② Assurer "l'équivalence" sémantique

Opérateur :

- ① $\oplus : \text{JCODES} \times \text{JCODES} \rightarrow \text{JCODES}$
- ② $\oplus_G : \text{JCODE} \times \text{JCODES} \rightarrow \text{JCODES}$
- ③ $\oplus_D : \text{JCODES} \times \text{JCODE} \rightarrow \text{JCODES}$

Liste neutre : jcnil

Compilation Déclarations



$$n + 1 \vdash dss \Rightarrow \{pdss, ndss\}$$

$$n + ndss + 1 \vdash mma \Rightarrow \{pmma, nmma\}$$

$$[\text{cclasse}] : \frac{n + ndss + nma + 1 \overset{\text{retrait}}{\vdash} dss \Rightarrow \{prdss, nrdss\}}{n \vdash \text{classe}(i, dss, mma) \Rightarrow \{init \oplus_G (pdss \oplus pmma \oplus prdss) \oplus_D \text{pop} \oplus_D \text{jstop}, ndss + nmma + nrdss + 3\}}$$



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



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



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



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

Compilation Déclarations (suite)

$$\textcircled{a} \quad [\text{ccst}] : \frac{n \vdash e \Rightarrow \{pe, ne\}}{n \vdash \text{cst}(t, \text{ident}(i), e) \Rightarrow \{pe \oplus_D \text{new}(i, t, \text{cst}, 0), ne+1\}}$$

$$\textcircled{a} \quad [\text{ctableau}] : \frac{n \vdash e \Rightarrow \{pe, ne\}}{n \vdash \text{tableau}(t, \text{ident}(i), e) \Rightarrow \{pe \oplus_D \text{newarray}(i, t), ne+1\}}$$

$$\textcircled{a} \quad [\text{centêtes}] : \frac{n \vdash en \Rightarrow \{pen, nen\}, n+n \vdash ens \Rightarrow \{pens, nens\}}{n \vdash \text{entêtes}(en, ens) \Rightarrow \{pen \oplus pens, nen+nens\}}$$

$$\textcircled{a} \quad [\text{centête}] : n \vdash \text{entête}(t, i) \Rightarrow \{\text{new}(i, t, \text{var}, k) \oplus_G jcnil, 1\}$$

$$\textcircled{a} \quad n \vdash dvs \Rightarrow \{pdvs, ndvs\}$$

$$n + nd \vdash iss \Rightarrow \{piss, niss\}$$

[cmain] :

$$\frac{n + ndvs + niss + 1 \overset{\text{retrait}}{\vdash} dvs \Rightarrow \{prdvs, nrds\}}{n \vdash \text{main}(dvs, iss) \Rightarrow \{pdvs \oplus piss \oplus (\text{push}(0) \oplus_G prdvs), ndvs+niss+nrds+1\}}$$

Compilation déclaration



$$n + 3 \vdash \text{ens} \Rightarrow \{\text{pens}, \text{nens}\}$$

$$n + \text{nens} + 3 \vdash \text{dvs} \Rightarrow \{\text{pdvs}, \text{ndvs}\}$$

$$n + \text{nens} + \text{ndvs} + 3 \vdash \text{iss} \Rightarrow \{\text{piss}, \text{niss}\}$$

$$[\text{cméthode}] : \frac{n + \text{nens} + \text{ndvs} + \text{niss} + 3 \overset{\text{retrait}}{\vdash} \text{dvs} \Rightarrow \{\text{prdvs}, \text{nrds}\}}{n \vdash \text{méthode}(t, i, \text{ens}, \text{dvs}, \text{iss}) \Rightarrow \{\text{jcnil} \oplus_D \text{push}(n+3)$$

$$\oplus_D \text{new}(i, t, \text{meth}, 0) \oplus_D \text{goto}(n + \text{nens} + \text{ndvs} + \text{niss} + \text{nrds} + 5)$$

$$\oplus \text{pens} \oplus \text{pdvs} \oplus \text{piss} \oplus \text{prdvs} \oplus_D \text{swap} \oplus_D \text{return},$$

$$\text{nens} + \text{ndvs} + \text{niss} + \text{nrds} + 5\}$$



$$[\text{cméthodeR}] : \frac{\vdots}{n \vdash \text{méthode}(t, i, \text{en}, \text{d}, \text{s}) \Rightarrow \{\text{jcnil} \oplus_D \text{push}(n+3)$$

$$\oplus_D \text{new}(i, t, \text{meth}, 0) \oplus_D \text{goto}(n + \text{nens} + \text{ndvs} + \text{niss} + \text{nrds} + 6)$$

$$\oplus \text{pens} \oplus \text{pdvs} \oplus \text{piss} \oplus (\text{push}(0) \oplus_G \text{prdvs}) \oplus_D \text{swap} \oplus_D \text{return},$$

$$\text{nens} + \text{ndvs} + \text{niss} + \text{nrds} + 6\}$$

Compilation Instructions

$$\textcircled{c} \text{ [cinstrs]} : \frac{n \vdash is \Rightarrow \{ps, nis\}, n+nis \vdash iss \Rightarrow \{piss, niss\}}{n \vdash instrs(is, iss) \Rightarrow \{pis \oplus piss, nis+niss\}}$$

$$\textcircled{c} \text{ [csomme]} : \frac{n \vdash e \Rightarrow \{pe, ne\}}{n \vdash somme(ident(i), e) \Rightarrow \{pe \oplus_D inc(i), ne+1\}}$$

$$\textcircled{c} \quad n \vdash e_1 \Rightarrow \{pe_1, ne_1\}$$

$$\text{[csommeT]} : \frac{n+ne_1 \vdash e \Rightarrow \{pe, ne\}}{n \vdash somme(tab(ident(i), e_1), e) \Rightarrow \{pe_1 \oplus pe \oplus_D ainc(i), ne_1+ne+1\}}$$

$$\textcircled{c} \text{ [cinc]} : n \vdash incrément(ident(i)) \Rightarrow \{jcnil \oplus_D push(1) \oplus_D inc(i), 2\}$$

$$\textcircled{c} \text{ [cincT]} : \frac{n \vdash e \Rightarrow \{pe, ne\}}{n \vdash incrément(tab(ident(i), e)) \Rightarrow \{pe \oplus_D push(1) \oplus_D ainc(i), ne+2\}}$$

Comp. Instructions (suite...)

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

$$\textcircled{a} \quad n \vdash e_1 \Rightarrow \{pe_1, ne_1\}$$

[caffecteT] :

$$\frac{n + ne_1 \vdash e \Rightarrow \{pe, ne\}}{n \vdash \text{affectation}(\text{tab}(\text{ident}(i), e_1), e) \Rightarrow \{pe_1 \oplus pe \oplus_D \text{astore}(i), ne_1 + ne + 1\}}$$

$$\textcircled{a} \quad n \vdash \text{lexp} \Rightarrow \{plexp, nlexp\}$$

$$[\text{cappelE}] : \frac{n + nlexp + 1 \overset{r}{\vdash} \text{lexp} \Rightarrow \{prlexp, nrlexp\}}{n \vdash \text{appelE}(\text{ident}(i), \text{lexp}) \Rightarrow \{plexp \oplus_D \text{invoke}(i) \oplus prlexp, nlexp + nrlexp + 1\}}$$

$$\textcircled{a} \quad n \vdash \text{lexp} \Rightarrow \{plexp, nlexp\}$$

$$[\text{cappelI}] : \frac{n + nlexp + 1 \overset{r}{\vdash} \text{lexp} \Rightarrow \{prlexp, nrlexp\}}{n \vdash \text{appelI}(\text{ident}(i), \text{lexp}) \Rightarrow \{plexp \oplus_D \text{invoke}(i) \oplus prlexp \oplus_D \text{pop}, nlexp + nrlexp + 2\}}$$

Comp. Instructions (...suite)

$$\textcircled{1} \quad [\text{cretour}] : \frac{n \vdash e \Rightarrow \{pe, ne\}}{n \vdash \text{retour}(e) \Rightarrow \{pe, ne\}}$$

$$\textcircled{2} \quad n \vdash e \Rightarrow \{pe, ne\}$$

$$n + ne + 1 \vdash s_1 \Rightarrow \{ps_1, ns_1\}$$

$$[\text{csi}] : \frac{n + ne + ns_1 + 2 \vdash s \Rightarrow \{ps, ns\}}{n \vdash \text{si}(e, s, s_1) \Rightarrow \{((pe \oplus_D \text{if}(n + ne + ns_1 + 2)) \oplus (ps_1 \oplus_D \text{goto}(n + ne + ns_1 + ns + 2)) \oplus ps), ne + ns_1 + ns + 2\}}$$

$$\textcircled{3} \quad n \vdash e \Rightarrow \{pe, ne\}$$

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

Compilation Expressions

$$\textcircled{c} \text{ [ctab]} : \frac{n \vdash e \Rightarrow \{pe, ne\}}{n \vdash \text{tab}(\text{ident}(i), e) \Rightarrow \{pe \oplus \text{aload}(i), ne+1\}}$$

$$\textcircled{c} \text{ [cident]} : n \vdash \text{ident}(i) \Rightarrow \{jcnil \oplus_D \text{load}(i), 1\}$$

$$\textcircled{c} \text{ [cnbre]} : n \vdash \text{nbre}(i) \Rightarrow \{jcnil \oplus_D \text{push}(i), 1\}$$

$$\textcircled{c} \text{ [cvrai]} : n \vdash \text{vrai} \Rightarrow \{jcnil \oplus_D \text{push}(jcvrai), 1\}$$

$$\textcircled{c} \text{ [cfaux]} : n \vdash \text{faux} \Rightarrow \{jcnil \oplus_D \text{push}(jcfaux), 1\}$$

$$\textcircled{c} \text{ [cop1]} : \frac{n \vdash e \Rightarrow \{pe, ne\}}{n \vdash \text{op1}(e) \Rightarrow \{pe \oplus_D \text{op1}, ne+1\}}$$

$$\textcircled{c} \text{ [cop2]} : \frac{n \vdash e_1 \Rightarrow \{pe_1, ne_1\}, n+ne_1 \vdash e \Rightarrow \{pe, ne\}}{n \vdash \text{op2}(e_1, e) \Rightarrow \{(pe_1 \oplus pe) \oplus_D \text{op2}, ne_1+ne+1\}}$$

$$\textcircled{c} \text{ [clexp]} : \frac{n \vdash \text{exp} \Rightarrow \{pexp, nexp\}, n+nexp \vdash \text{lexp} \Rightarrow \{plexp, nlexp\}}{n \vdash \text{listexp}(\text{exp}, \text{lexp}) \Rightarrow \{(pexp \oplus plexp), nexp+nlexp\}}$$

Comp. Retrait des déclarations

- ③ $[\text{crexnil}], [\text{renil}], [\text{rvnil}] : n \vdash^r \text{vnil} \Rightarrow \{jcnil, 0\}$
- ③ $[\text{crvar}] : n \vdash^r \text{var}(t, \text{ident}(i), e) \Rightarrow \{jcnil \oplus_D \text{swap} \oplus_D \text{pop}, 2\}$
- ③ $[\text{crcst}] : n \vdash^r \text{cst}(t, \text{ident}(i), e) \Rightarrow \{jcnil \oplus_D \text{swap} \oplus_D \text{pop}, 2\}$
- ③ $[\text{crméthode}] :$
 $n \vdash^r \text{méthode}(t, \text{ident}(i), en, d, s) \Rightarrow \{jcnil \oplus_D \text{swap} \oplus_D \text{pop}, 2\}$
- ③ $[\text{crdecls}] : \frac{n \vdash^r dss \Rightarrow \{prdss, nrdss\}, n+nrdss \vdash^r ds \Rightarrow \{prds, nrds\}}{n \vdash^r \text{decls}(ds, dss) \rightarrow \{prdss \oplus prds, nrdss+nrdss\}}$
- ③ $[\text{crvars}] : \frac{n \vdash^r dvs \Rightarrow \{prdvs, nr dvs\}, n+nrdvs \vdash^r dv \Rightarrow \{prdv, nr dv\}}{n \vdash^r \text{vars}(dv, dvs) \rightarrow \{prdvs \oplus prdv, nr dvs+nrdv\}}$
- ③ $[\text{crlexp}] : \frac{n \vdash^r \text{lexp} \Rightarrow \{prlexp, nrlexp\}}{n \vdash^r \text{listexp}(e, \text{lexp}) \rightarrow \{jcnil \oplus_D \text{swap} \oplus_D \text{pop} \oplus prlexp, nrlexp+2\}}$

Exemple

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

Exemple (suite...)

Code MiniJaja	Adresse	JajaCode
}	13	swap // retrait de y
	14	pop
	15	swap // adr retour ↔ val. résultat
	16	return
void 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)
	23	invoke(f)
	24	swap // retrait du paramètre
	25	pop
	26	inc(y)

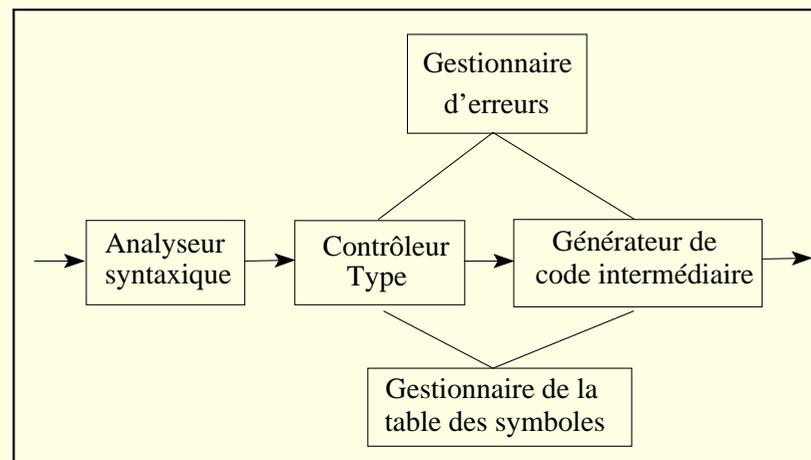
Exemple (...suite)

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

Contrôleur de type

Le système de typage : Statique

- ③ *Contrôle de type* : incompatibilité de type des opérandes.
- ③ *Contrôle du flot d'exécution* : transfert du contrôle en un autre point du programme.
- ③ *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 = **Spécification du langage**

Expressions de type :

- ① *Type de base* : Booléen, Caractère, entier, réel, void...
- ① *Noms de type*
- ① *Constructeurs de type* :
 - Tableau
 - Produit
 - Enregistrement
 - Pointeur
 - Fonction...
- ① Variables pour définir des *expressions de type*.

Les systèmes de typages

- ① Mis en œuvre par le contrôle de type
- ① Choix par le concepteur du langage (Ada, Pascal, C...)
- ① Éléments externes pour le débogage : lint
- ① Équivalence de type : Structurelle, nominale

Contrôle statique vs dynamique :

- ① Langage
- ① Éléments seulement connus pendant l'exécution

Récupération des erreurs :

- ① Technique de récupération identique à l'analyse syntaxique.

Typage est 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

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 leur sémantique 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.

Définition 1 (Équivalence) :

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 :

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

On le note $equiv(m, m', i)$.

Définition 2 (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

$\forall PN \in L, \forall m \in MEM, \forall i \in ID :$

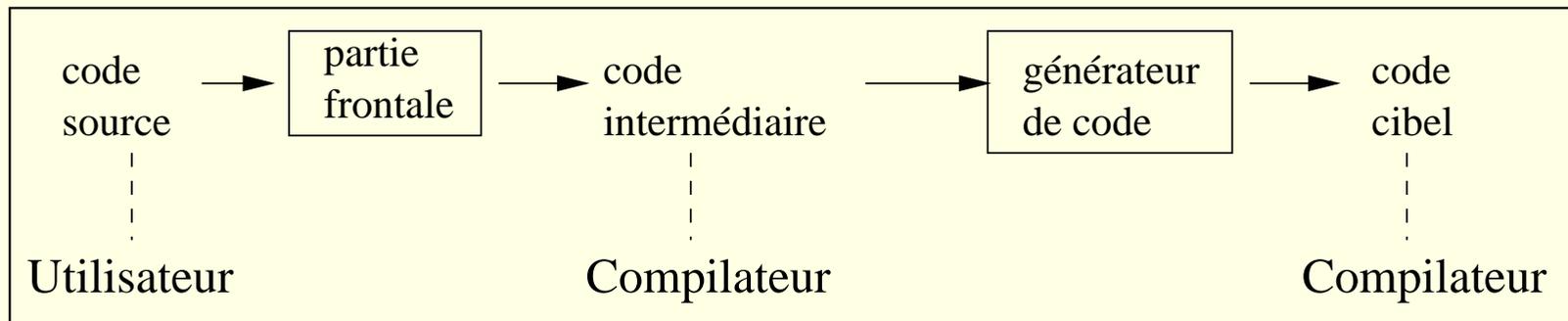
- ⊙ $S_{opL}(PN, m) = m' \wedge S_{opL'}(c[PN], m) = m_0, m_1, \dots, m'' \Rightarrow \text{équiv}(m, m', i)$ C'est à dire :
- ⊙ $m \vdash \Rightarrow m'$,
- ⊙ $a \vdash PN \Rightarrow \{PC, n\}$
- ⊙ $\langle m, a \rangle \vdash PC \leftarrow \langle m'', a' \rangle$

Optimisation de code

Critères :

- ① Préservation de la signification du programme
- ① Accélérer les programmes
- ① Rapport coût / gain intéressant

Il existe trois niveaux d'optimisations :





Optimisation

Utilisateur :

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

Compilateur :

Code intermédiaire :

- Boucles
- Appels de procédures
- Calculs des adresses

Code Cible :

- Utiliser les registres
- Sélectionner instructions
- 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 :
 - (a) La première instruction
 - (b) Toute instruction atteint par branchement
 - (c) Toute instruction qui suit un branchement
2. Un bloc correspond aux instructions commençant par un bloc de tête jusqu'au 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, \dots, n_k les fils de n ordonnés
par Étiquette décroissante

Étiquette(n) \leftarrow $\max_{1 \leq i \leq k} (\text{Étiquette}(n_i) + i - 1)$

Fin

Fin