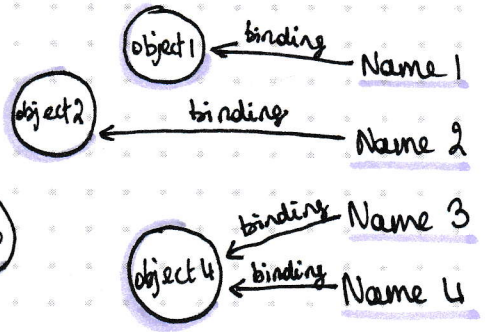
CONSTRUCTION DES COMPILATEURS

[...]

Scan : token → Parser : OK/KO grammar + AST → désucrage syntax° → analyse semantique

Analyse semantique : noms, scope, binding.

Nom = symbole. Un symbole pointe sur une zone mémoire.
alphanumeric et underscore, commencé par une lettre, sans whitespace.
limite de taille ⇒ FORTRAN : 6 l. C : 31 l Autres : no limit
Case sensitive sauf dans quelques lgg comme Modula-2 et ADA



Name 1 = object 1 {}
Name 2 = object 2 {}
Name 3 = object 3 {}
Name 4 = object 4 {}
Name 3 = Name 4

→ Il faut aussi penser à gérer les alias!
(On s'en tout en fait tant qu'on arrive à remonter à l'origine)

Quand est-ce que sont créés et détruits | les objects ? Lifetime
les noms ? Scopes
les alias ? Binding times

Scopes : { }, statique, class & struct, namespaces, fonction, nom de fonction, template, bloc conditionnel, enum

→ 1st forme de structure et modularité
→ Sans scope : influence globale. Avec scope : influence locale
→ Dans un pgm avec des id uniques, les scopes sont inutiles

No scope in Assembly & MFS (Mac Intosh File & the 1st)
⚠ Deux scopes peuvent se croiser! int x = ... ; { int x = x + 42; }

Static scoping : typage fort → + clair et + rapide

Instant de liaison :
- lgg design
- lgg implem
- pgm writing
- compilation
- linkage
- loading
- execution

the Moving IN

early ——— binding time ———→ late
- inflexibility ……→ + flexibility
+ efficiency ←……- inefficiency

(change mode)
→ VMCS

Table de symbole : | sym | val |

- traversal check uses against def
- form of memory
- related to scope
- reversible memory

Associative array → put , get
Implem : • list • tree • hash • ...

Complications :
→ Overloading
→ Escape
→ Functionnal pgm

TYPES

Pourquoi utiliser des types ?
- Controle + tôt des choses
- Echapper au paradoxe de Russel : $E = \{x \notin x\}$ $E \in E$ $E \notin E$

Les types ne sont pas nécessaires (assembler, tcl, Mli...)
Mais ils sont utiles
→ optimisat°
→ controle
→ abstract°
→ managemt de la mémoire

Type checking : les opérat° sont-elles valides ?

Types : numeric, booleens, énumérat°, sous intervalles, tableau (+heap+stack) unions, structures / objets, tuples / listes, reference / ptr, ...

Coercion : conversion implicite d'un type vers un autre.
- widening : petit vers grand (ex : int → float)
- narrowing : grand vers petit (ex : float → int) ⚠ Perte d'informat° !

Lgg fortemt typé : erreurs tjs detectée (runtime ou compilat°)
→ aucun risque (mais pénible pr utilisateur)

2 types sont équivalents si une opérande d'un type dans une expr est substituée par l'autre type sans coercion.

$(e_1 : int \wedge e_2 : int) \Rightarrow e_1 + e_2 : int$

Regles d'inference : $\frac{\vdash Hyp_1 \dots \vdash Hyp_n}{\vdash Cl°}$

$\frac{\vdash i \text{ is an int}}{\vdash i : int}$ $\frac{\vdash e_1 : Int \wedge \vdash e_2 : Int}{\vdash e_1 + e_2 : Int}$ (Pour le reste cf LOFO)

In Tiger: $\frac{}{\Gamma \vdash n : \text{Int}}$ $\frac{}{\Gamma \vdash s : \text{String}}$ $\frac{}{x_1 : A_1, \ldots, x_n : A_n \vdash x_k : A_k}$ $\frac{\Gamma \vdash a : \text{Int} \quad \Gamma \vdash b : \text{Int}}{\Gamma \vdash a + b : \text{Int}}$ +

$\frac{\Gamma \vdash c : \text{Int} \quad \Gamma \vdash t : A \quad \Gamma \vdash f : A}{\Gamma \vdash \text{if } c \text{ then } t \text{ else } f : A}$ if then else

(dark mode)
↳ VMCS