Le front-end

NB. J'ai écrit tout ce que le prof disait donc pas forcément tout est « important » dans le cours.
Ce qu'il faut pour l'examen :

Être capable de prendre un langage : comment on fait pour rajouter des options ?

- Parser/scanner (les endroits à modifier) : reflexion derriere ca
- Etre capable de Construire un ast – visiteur
- Binding : être capable de lier
- Time shacking

Cours 1:

- Qu'est ce qu'un compilateur ?

Traducteur d'un langage vers un autre.

1. Pre processing (cpp)
2. Compilateur (cc1) >> Tiger
3. Assembleur
4. Linkage (static or dynamic)

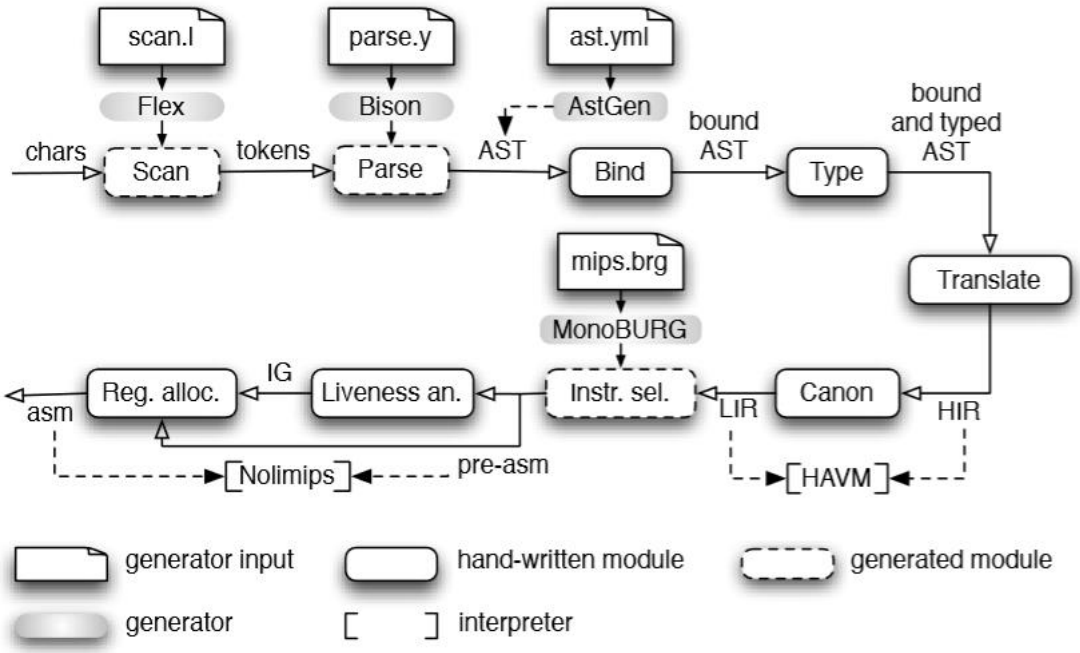


Schéma des étapes de compilation

A la base ya un fichier (cf flex et bison) scanner remplace ce qu'il connait pas un token (if, else, while, etc)

Le parser doit interpréter les différents tokens (construction de l'ast).

Le binding (a + b d'où vient le a d'où vient le b/ scoping).

Instruction sélection (plein de choix possibles,..).

Attention il faut tester à chaque fin de module.

NB. HAVM = machine virtuelle écrite en Haskell = exécuter du code de haut niveau (utile pour tester).

A retenir l'idée d'un pipe > on passe d'une étape à une étape. Qu'est ce qu'on passe d'ailleurs ?

Compilateur large (un fichier)

Compilateur étroit (une fonction)

> ex. GCC (jusqu'en 2006 pourquoi il est resté aussi longtemps en mode étroit ? > la mémoire, était trop faible à l'époque, quand ya eu des mémoires à plusieurs méga il est passé en mémoire large).

Y'a-t-il d'autre stratégies de compilation ? Oui. Ya des compilateur qui s'arrête à canon

(cf. Java qui donne la partie basse à une vm) >> tout ce qui est à la base de JVM, .net > **bytecode strategy**.

Hybrid approches (on fait couche par couche comme un oignon).

Modular systems (on peut faire module par module)

Tiger exemple. print(« Hello World ») ;

- Ca implique qu'il y une runtime, des fonctions qui ne fonctionnent pas exactement comme les autres. Gestion de fonctions.

Il y a la notion de bloc.
Ya aussi de la gestion de typage pour être capable de définir quelle fonction on appelle si ya un int il faut tel ou tel fonction. Dernière variable calculée = variable retournée.
Attention il y a aussi de la récursion > il faut donc comprendre la pile.

Scanner and Parser

Qu'elle est l'utilitée d'un scanner et un parser ?

- Pas la sémantique
- Il y a une arborescence flagrante même si c'est un texte linéaire. Transformer quelque chose de linéaire en une arborescence.

Flex = lexical analyser

= flux de caractères et faire une analyse lexical (ex : ça c'est un identifiant de fonction, a c'est une lettre, 0 c'est un nombre, if c'est un identifiant etc). Permet de manipuler 4 concepts (yytext, yyleng, yylex, yywrap). Flex on lui définit un fichier de type structure (cf. slide) et il génère un fichier en c.

- Partie haute = définition de ce dont j'ai beosin avant du fichier.
- Yylex = stp est ce que tu peux commencer à lexer >> Scanner. Génere le fichier qui va bien.

- Yywrap = ? (execute le programme en C).

Bison = le parser

Le parser prend des tokens en paramètres. On utilise Bison (attention bien utilser le Bison du site du lrde). Bison ressemble pas mal a la syntaxe de flex. Bison va générer un parser (un fichier) > un flux de token en entrée, ceci correspond a la grammaire que tu m'as défini. De base bison fait du LALR.

Différents types de conflits ?

LALR > **Reduce/reduce** = je définis un ensemble de règles si on voit if/then/else = on cree un bloc, on réduit on merge le tous. **Shift/reduce** = est ce que je dois faire une réduction ou juste lire le mot d'après ?

GLR > si ma grammaire est ambiguë, j'ai un conflit= je vais essayer les deux directions en même temps, si je tombe dans un cul de sac c'est que c'est l'autre.

C'est ici qu'on repère les syntax error (le fichier ne match pas notre grammaire, c'est pas bon).

Pb du dangling else. A quel if on appartient ?

Attention bien mettre un %except 0 (on veut 0 conflicts)

%right « else » « then » (%right : choose shift

%left: choose reduce)

- En gros il faut prendre une solution, en cas de conflits on guide bison pour qu'il choisisse

Slide exemple 3 ambiguous grammar :

- Ca fait des tableaux. Si on a un id, on sait pas trop si on est dans une lvalue ou dans un typeid (on peut essayer de remonter le typeid, car il sert pas vraiment, voir il sert a rien, id il peut nous envoyer dans deux règles différentes..). On sait pas ou on doit aller.
- Concrètement on peut soit faire une déclaration de tableau (shift) ou soit faire une lecture de tableau (reduce)

Solution (attention il manque %except 0)

- Trois règles plus compliques.. quand ya une grammaire ambigue il faut la retravailler.
- Il faut parfois décomposer nos règles pour les décomposer. Ca va bien se passer.

Coupling flex et bison (cf steps dans les slides).

Mais pb : si on rencontre un identifiant, il faut donner l'identifiant a bison. Donc on utilse les chevrons >> indiquer qu'un token veut une valeur.

Api.token.prefix = pour éviter les conflicts.

Bison $$ = valeur de résultat de la règles.

$1 premier elem de la règle ; $2 deuxieme elem de la règle ; etc.

Un parser = valeur en entrée et on descend dans les règles jusqu'à trouver une règle feuille.

The parser maintains a stack of type = In C no pb c'est du C on utilise une union. (void)

En C++ >> on utilise les variants. (pour régler le problème de destructeur, qui sait pas quel type il doit détruire).

Attention !! On est pas obligé d'utiliser des variants.

Cours 2 : 06/02/2019

%token <int> INT (tokens à qui on attribue un identifiant)

%token <misc ::symbol> ID (éviter une redondance mémoire/saturation de la mémoire)

Attention : ne pas rentrer en conflit avec d'autres variables qui existe etc.

Compilateur son role 1 : traduire

Role 2 : vérifier que c'est correct – si ya plus de règles le compilo dit ya une erreur, il faut quand même qu'il continue ; minimiser le temps de la compilation (gros projet = bcp de perte de temps)

Mais comment on fait pour continuer la compilation ?

- Ou est ce que je m'arrête ? on lit les tokens les uns a la suite des autres et on va dismiss tout les tokens qui nous conviennent pas jusqu'à ce qu'on tombe sur un token qui nous convient. Mais probleme des erreurs en cascade. On fait du Best effort.
- Ensuite on prend tous les tokens qu'on a skip = qu'est-ce qu'on fait ? on les remplace par une erreur

Il faut etre capable de tracer la ou il ya une erreur : on fait dans flex avec des locations :

- Initial action
- Yy user action : loc.columns In the parser
  Loc.step()
  Loc.line

  Loc c'est un objet qui fait de la gestion pour nous.

@ = position renvoyer dans flex par notre règle.
@$ = position du resultat (cf la doc).
Location > c'est flex le mieux placer pour le trouver. Donc flex est en charge de l'envoyer a bison (bison il fait le lien facilement avec les locations de flex en utilisant @ et %locations)

Improving Scanner et Parser

En quoi ca pause un pb que le parser connaisse notre AST ? ca implique qu'on a une dépendance entre l'ast et le parser. Quand notre ast est modifier on doit modifier le parser (or c'est super chiant).

Donc on va faire une Factory = Driver (une petite classe entre le parser et l'ast, on pourra modifier l'ast sans toucher au parser). Tyger-parser.yy = class TigerParser.

Quel pb entre stdin et un fichier ?

- Fermeture des fichier

Pb avec les #include !!

- Pb ca casse l'analyse du fichier : on fait un token, et on parse le fichier
- Il faut parser l'import !

Tr = trans (ca modifie tous les trucs)

Awk = colone 1 et colone 2 et colone 3 d'un fichier

GCC's C parser = gcc 3.2 était écrit en Bison

Development Tools

Les modules on veut qu'ils soient indépendants en C++ > on fait des bibliothèques.

Ecrire un main = permet de faire le lien entre les modules.

Sinon on utilise le design pattern Command : on va avoir une bibliothèque independante mais un composant aura un effet de bord (c'est ce qu'on appelle une task).

Autotools ça fonctionne comment ?

- Si on veut faire un vrai projet on est obligé de gérer tous les types d'ordis. Les built systèmes permettent d'avoir tous les systèmes d'exploitation

    1. Une sonde est lancé sur l'ordi (c'est quoi la version de gcc que t'utilise etc. est ce que tas un mac ? je vais patcher ca pour que tout se passe bien)
    2. Il faut coupler autotools avec automake : on va demadner a l'utilisateur de faire un makefile générique (meme syntaxe, mais les dépendances entre les modules etc.
    3. ./configure install etc.

Boost/bison/flex/gettext/libtool/automake/…

C++ = on peut implémenter k, et faire un truc haut niveau en même temps. On peut faire un grand écart.

Bindings = permet d'utiliser une lib de C++ en python.

(ca parle de autotools, automake etc) // difference CMake/Autotools
« autotools c'est mieux » en gros. = il faut mettre les mains dans le pâté.

Tools for Developer

C++ ⇔ rigueur ! Utiliser des Warnings (CXXFLAGS)
Assert macro == on peut les désactiver quand on est en dev.
Clang tidy/clang format = passer les moulinettes des assistants.
LLVM = clang static analyzer
Gprof, OProfile :
Comment on fait pour améliorer la rapidité de ton code ? (-o1/-o2/-o3)

- Profiler ça sert à dire : tu passes 10% de ton temps dans cette partie de ton code. Problème c'est un peu lent donc faut le faire sur des petites parties de code.

(NB. Lire du code : d'abord le main, puis chaque fonction appeler scruté)

Abstract Syntax Trees

Cours 3 : 08/02/19

Lexer scan + envoie des tokens >> Le parser récupère la grammaire et dit c'est bon ou c'est pas bon ?

>> si c'est bon on veut faire un AST.

Pourquoi on a besoin d'un AST dans tiger ?

- Context/scope (ouais mais non)
- Les branchement/boucle (ouais mais non)
- Pouvoir définir des fonctions et des variables ! OUI

Les variables font qu'on a besoin d'un ast.

Identifiant + identifant ? comment on fait ? (une hashmap .. ouais mais plusieurs a possible dans différents scopes.. laquelle on prend)

Arbre de syntaxe concrète = ya des parenthèses.

Arbre de syntaxe abstrait = ya moins d'informations dedant. (cf. slide : a droite c'est abstrait)

En Lisp = sa syntaxe abstraite est la meme que la syntaxe concrète.

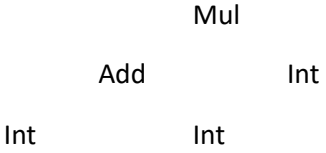
Un compilateur est un looong pipe, chaque module doivent échanger de la donner (un flux de tokens).

Est-ce qu'on écrit l'ast a la mano ? Comment on fait pour faire communique les différents modules ensembles.

- Stocker l'ast dans un fichier XML lu par tout le monde, mais c'est lent et ca nous fait aussi ouvrir des fichiers/fermer des fichiers == lent. + ca nous fait faire plusieurs parsers (un pour chaque module)
- ASN 1 = alternative a XML. On définit une interface pour dire au receveur comment il va recevoir la donnée.
- Protobuff baser sur ASN 1. Définit un squelette dans un langage donné.
- CORBA, JSON, YAML (pour générer du C++), SDL, etc.

Comment on va faire proprement ? design pattern command. Variable partagée de manière propre. On utilise une petite partie impure.

Un ex : Pour le problème suivant :



On peut faire quatre classe : une classe Expression une classe BinopPlus BinopMult une classe Num

(Trois classes c'est ok aussi).

Traversals in compilers:

- Pretty printer
- Name analysis
- Unique identifiers (attention important cf inlining)
- Desugaring (for >> while >> go to >> assembleur c'est le but)
- Type checking
- Non local (escaping) variable:
  Quand je fait &i est ce que ca interesse le compilateur ? I il est dans un registre. Variable traité spécifiquement. Variable en échappement (devrons être traité séparément)
- Inlining (conflit de nommage possible…) on remplace la déclaration directement par le code.
- High level optimizations : je vais intervertir les boucles for
- Translation to other intermediate representations
- Etc.

Pourquoi on peut pas faire les phases en meme temps ? On peut name analysis et unique identifiers.

Donc oui on peut mais il faut savoir comprendre les 8 étapes. (cf le livre de A. Appel)

But de l'ast :EFFICACE et Propre/ minimaliste. Parce que bcp de traversées.

Slide : 28/105 (code review) (pb d'affichage, type de classe, etc.)

- Le probleme vient du C++ : concept de type static/type dynamique
  Ex : Bin *bin = new Bin(…) ; Type static = Bin type dynamic = apres le new.
- Le probleme ici = c'est un probleme de dispatch.
- Dynamic cast / ou alors on va faire du virtuel => mais on peut pas sur l'opérateur >> ? Non
- Bref la en gros il explique l'override c'est cool pour le printer etc. C'est stylé de ouf.
  (comment faire un pretty printer)

NB. Sur un ast combien de classes ? 30/40/50 sur un langage comme Tiger -- Une centaine de classes sur du C.

Le probleme avec l'override de print = c'est qu'il faut ajouter des prints dans **chaques classes**. Mais ya un truc c'est que print ca va encore pour le debug. Mais quand on aura d'autre pb il faudra tout ouvrir pour debug >>> ca c'est chiant.

Donc comment on fait : on commence par décomposer les choses = il faut faire un dispatch = utilser une fonction print qui fait JUSTE le **dispatch**. On separe les concepts :

- Traitement
- Dispatch

Techniques super stylée « plus grande arnaque du siècle » **Multimethods**

o.foo(o1, o2) ; on appelle foo sur o avec comme argument o1 et o2. Si foo est virtuel on va faire le dispatch pour trouver la bonne implémentation. O a un type dynamique et au runtime tu vas utiliser le type dynamique au runtime pour savoir quelle fonction utilisée.

Si on fait foo(o, o1, o2) ; pb on a un langage de faible on se rend compte qu'il dit je peux faire le dispatch dynamique sur le premier argument uniquement ! Si on avait pas ce problème là, on aurait plus de soucis. C'est le probleme du C++. No multimethods en C++ (Lisp c'est cool).

Donc a un moment donné on va devoir trouver une solution élegante pour demander à la hiérarchie de faire le dispatch pour nous.

Cf slide (code review)

La fonction dispatch prend deux fonctions en paramètre qui retourne du void. Dans Bin il faut implémenter le dispatch (revoir les lambda fonctions).

MAIS pb = deux inconvénients majeur :

1. On a un call back (tu m'appelles plus tard quand tu en sais plus).
2. Un call back par classe (une vingtaine de parametres sur la fonction).
3. Probleme d'utilisation : les gens ont le droit de l'utiliser mais pas de modifier la classe. Donc pas du tout extensible.

Donc il nous faut une meilleure solution encore.

"If you have a procedure with ten parameters, you probably missed some." > autrement dit c'est de la chiasse.

0n va donc remplacer dispatch par une classe. (Ask the hierarchy to Dispatch)

Plus ou moins de call back en fonction du nombre de classe dans l'ast.

Cours 4 : 11/02/2019 : Solution au pb : Classe Visitor

(slide 53/105)

On veut des arguments dans les call back.

Maintenance assez chiante et ca peut commencer a prendre un peu trop d'arguments (ca va finir par saturer la pile)

On a une fonction avec des fonctions et des données >> on a envie de faire une classe.

C'est le design Pattern VISITOR.

Ici le pb auquel visiteur répond c'est :

Comment faire un dispatch dynamique en étant exclusif.

(virtual .. const = 0 >> virtuelle pure) On va utiliser cette classe exactement de la meme manière que la fonction call back.

Pretty printer devient un visitor (hérite de Visitor).

Parcours desucrage = Visitor a le droit de modifier l'AST (iterator = visitor)

PrettyPrinter parcours = n'a pas le droit de modifier l'ASR (const_iterator = const_visitor)

ATTENTION : un objet en mémoire = juste les arguments.

Conseil = on prefere clarifié le code et perdre un peu en efficacité.

DefaultVisitor DefaultConstVisitor = parcours a vide de l'ast. On fait hériter une classe pour avoir un comportement specifique pour un nœud donné. Cf slide 72 > Visitor Combinators. On monte en abstraction > les choses sont plus claires.

Théorie des lambdas calculs

Désucrage = simplification

- Context switch = fait par le compilateur : clarification. (Compilo a langage fonctionnelle) gcc faut pas ca/
  On met toutes les fonctions a plusieurs arguments a un argument.

- Let in : deviens une fonction
  Etc.
- Cf desucrage syntaxique pour quick sort

Tous les langages font du desucrage.

Desucrage : un exemple

If a then B ➔ if a then B else C

Qu'est- ce qu'on met pour C ?

> on peut mettre un 0 ou un false ? MAUVAISE idée

> on doit trouver un truc qui fais pas d'effet de bord void.

Names, scopes and bindings

Scope = c'est un ensemble de définition

{ } et aussi les fonctions, les if ..then .. else les namespaces. ➔ différents niveaux de visibilités. Gestions des identifiants.

Scoping

Static Scoping > compute at compil time (quand est ce que ma variable, mes symboles sont disponibles).

Dynamic Scoping > depends on runtime conditions such as function calls.

- Le bash à du dynamic scoping par exemple

Cours du 18/02/2019

A quoi ça sert les scopes ?

- Pour avoir de la modularité : plus obligé de connaitre tout mon programme pour écrire du code, je peux écrire des bouts de code local
- Attention en assembleur par exemple, il n'y a pas de scope ➔ pas obligatoire mais ajoute une fonctionnalité.

Scope global = tout ce qu'il y avait avant moi et qu'il y aura après moi.

Slide 12/56 : outer renvoie une fonction.

(revoir les variables en échappement).

Typage fort ?

- C ➔ void * donc non
- C++ ➔ typage dynamique
- Ada, Tiger ➔ langage fortement typé

Slide 15/56 : combien y a-t-il de t défini ? bcp lol

Combien de scope en C++ ?

- Scope global
- Fonction
- Namespace
- Classe
- Template
- (lambda fonction)
- Enumération
- Les accolades ({ .. })
- Les variables définies dans le if (ICI) { .. } else { .. } et elles seront définies dans le else aussi.

On définit des environnements qui correspondent aux différents scopes (cf. slide avec des sigmas).

Instant de liaison. (Binding time)

On a des liaisons entre un symbole et un comportements insatnt de liaison c'est if ➔ lien avec un scope etc.

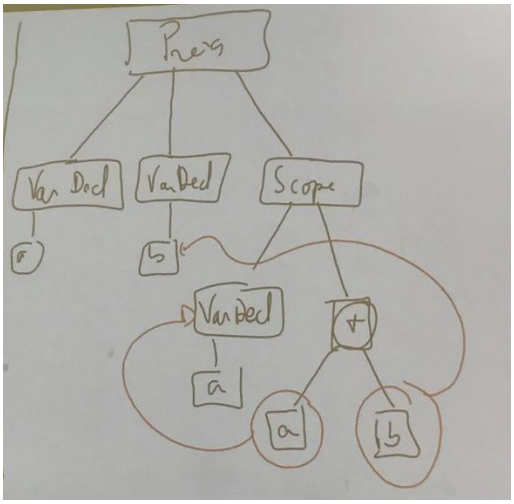
Il existe plein d'instants de liaison = ca serait cool si tout le compilo faisait ces instants de liaison/ ou alors on le fait au run time.

Pourquoi faire les instants de liaison plus tot ?

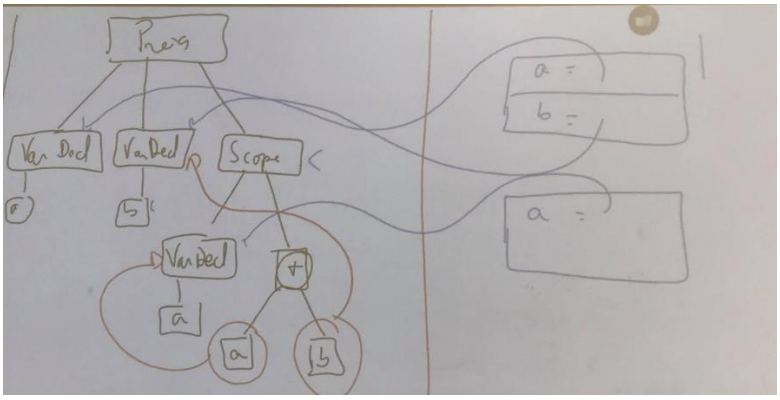
- Ca permet de faire de l'optimisation. (en ML) mais langage pas tres flexible. Python = instant de liaison très tardif donc langage plus flexible.
- Perl et Python plus flexible que C++ et ML par exemple.

Comment on va gérer nos scopes ? **Table des symboles**

- Hash table qui stock tous les symboles = c'est une idée de merde.
  Parce que il faut pas avoir deux définitions du même noms pour les symboles
  Parce que ca revient a n'avoir qu'une seule scope
- Idée : maintenir un environnement = une liste ou une hashmap. Avec des pointeurs sur les definitions.



Inconvénient = le pb c'est que si il y a plein d'appelle de b et plein de scope il faut remonter tous les environnements donc c'est pourri. Couteux en temps.
Donc on pourrait recopier le dico dans chaque scopes. C'est simple mais c'est couteux si ya plein de donnée (mémoire). Couteux en mémoire.



Comment il fait GCC ? ni l'un ni l'autre

- Il a une hashtable le symbole pointe vers le bon scope (si a est redéfinit on change de pointeur ! Attention a bien garder un pointeur vers l'autre scope)
- Il maintient une liste : qui tient toutes les variables définit dans le scope courant avec un marqueur qui délimites les différents scopes. Et après il supprime les éléments de la liste du scope courant (en utilisant les marqueurs) ! Attention a la suppression a la mise a jour de la hash table

Binder

Super Pouvoir du while et du for : break = c'est les seuls qui ont le droit d'utiliser break et continue ; ce break et ce continue on le traite au dernier moment dans la chaine de compilation (le binder doit lier le break au bon while ➔ pb si ya plusieurs while imbriqués !)

Overloading is syntactic sugar : mangling convention de language

La récursion et les scopes : comment on gère ça ? C'est la mer noire.

- Lien lié statiquement
- Durée de vie de la variable
- Savoir quel bloc d'activation je suis.

Cours du 18/03

Cf **règle d'inférence**.

https://fr.wikipedia.org/wiki/R%C3%A8gle_d%27inf%C3%A9rence

Tous ce qui est phénome de coercition = je pars d'un type large et je le rétréci ou je l'agrandis on peut mettre l'opérateur <=

Problème de typage : if then a else b. Si b est de type float et a de type int (supertype float) alors la valeur de retour est celle du supertype. Si pas de supertype = erreur de typage.

On cherche si pas d'erreur de typage on cherche le plus petit ancêtre commun a a et b :

lub(X, Y) = Z (Z est l'ancêtre commun le plus proche) entraine :

- X <= Z et Y <= Z
- Si X' <= Z' et Y <= Z' alors Z <= Z' (<= == « est une sous classe de » c'est un symbole pour comparer des types)

Pour If c then a else b qu'elle est la règle d'inférence :

|- C booléen |- a :X |-b :Y

|-if c then a else b:lub(X,Y)

(derriere les deux points c'est le type)

Théorème de robustesse : Type statique (avant compil) >= type dynamique (à la compil).

Le type statique est englobant. Un enfant aura tjs plus que ces parents.

Comment on implémente <= ? Qui va nous servir a définir toutes nos règles d'inférences. Celle qui peuvent être définit par l'utilisateur et celle qui existe.

Def <=(X, Y) :

    If X is int and Y is float

        Return (true, float)

Pour tous les types il faut faire ça !! mais si l'utilisateur veut ajouter un type : ça marche plus… c'est nul.

Meilleure solution :

Il y a deux catégories de type : types primitifs et types utilisateurs (classes).

<=	Classes	Type primitif	Tableau	Null	Error
Classes	Yes si il y a un ancêtre commun	No	No	No	No
Type primitif	No	Yes si les types sont compatibles	No	No	No
Tableau	No	No	Yes si deux les types des tableaux sont des types compatibles (et la taille ?)	No	No
Null	Yes (null <= A)	No	No	yes	No
Error	Yes	Yes	Yes	Yes	Yes

Nb. Pour les tableaux : est-ce que int tab[8] et int tab[12] sont de même types ? Certains langages disent non mais c'est pas une bonne idée c'est très contraignant par exemple pour les fonctions (il faut une fonction pour les tab int[8] une autre pour tab int[12])

Nb. Error est un type plus petit que tout le monde

NB. Système de typage très propre (avec un objet le plus grand Objet (enJava) et un objet le plus petit Error.

NB. Un pointeur c'est quoi ? en fonction des langage : un pointeur peut etre un type primitif (en Pascal). En C++ et en C c'est un peu ambiguë.

En fait pour les fonctions qui sont des types, void, les pointeurs, on va faire des nouvelles colonnes en fonction de ce que l'on veut implémenter dans notre langage.

Overloading

f(Base, Base) ok

f(base, Derived)

f(Derived, Base)

avec **Base ← Derived**

Quel appel va nous mettre en difficulté en tant que compilateur ? Si on fait comme appel : f(Derived, Derived) le compilateur est incapable de choisir laquelle des trois prendre il a une préférence pour les deux dernières fonctions mais il est incapable de choisir. Donc la le compilateur il peut dire : c'est ambigüe. Stop !!

On l'implémente comme ca.

On rajoute une fonction variadique : f(Base, …)

1) va faire une hiérarchie de fonction : de la plus générique à la moins générique. On groupe par niveau, il n'y a pas d'ordre.

F(Base, …)

|

F(Base, Base)

/ \

F(D, B) F(B,D)

2) On commence par le bas de la hiérarchie on filtre toutes les fonctions :

- Si j'en ai 0 je remonte
- Si j'en ai qu'une je remonte
- Si j'en ai deux je suis ambiguë

Le runtime

Il faut que je sois spécifique à chaque architecture quand je fais du runtime. Donc je veux que ce soit le plus petit possible.

Le compilateur doit gérer la runtime qui est spécifique à chaque langage.

On veut donc restreindre le langage avec des **atomic builtin** : **Logical** (Boolean) les **numerical** (Integer, Float, Fixed, Complex, tc) et les **character**. Et les **user specific** vont se comporter comme les builtins.

Comment on implémente une chaine de character :

- Un tableau avec un \0 à la fin. Mais on peut pas avoir la taille en temps constant.
- Une liste chainée : insertion où on veut, réduction quand on veut ou on veut. Mais pas très optimal. Mais tjs le problème de la taille. Problème accéder à un élément c'est la mort.
- Faire une structure : qui contient la taille etc.

Comment on fait la runtime pour une énum :

- Un énumération ça peut être un int. Mais pourri parce que on peut comparer des int alors qu'ils ne sont pas des int.