CEI 2015 – Compilateur 4D – LLVM
Proof of concept

Florian Lemaitre & Aubry Herve

Superviseurs Supélec : Gianluca Quercini & Idir Aït Sadoune

Superviseur 4D : Damien Gérard & Laurent Esnault

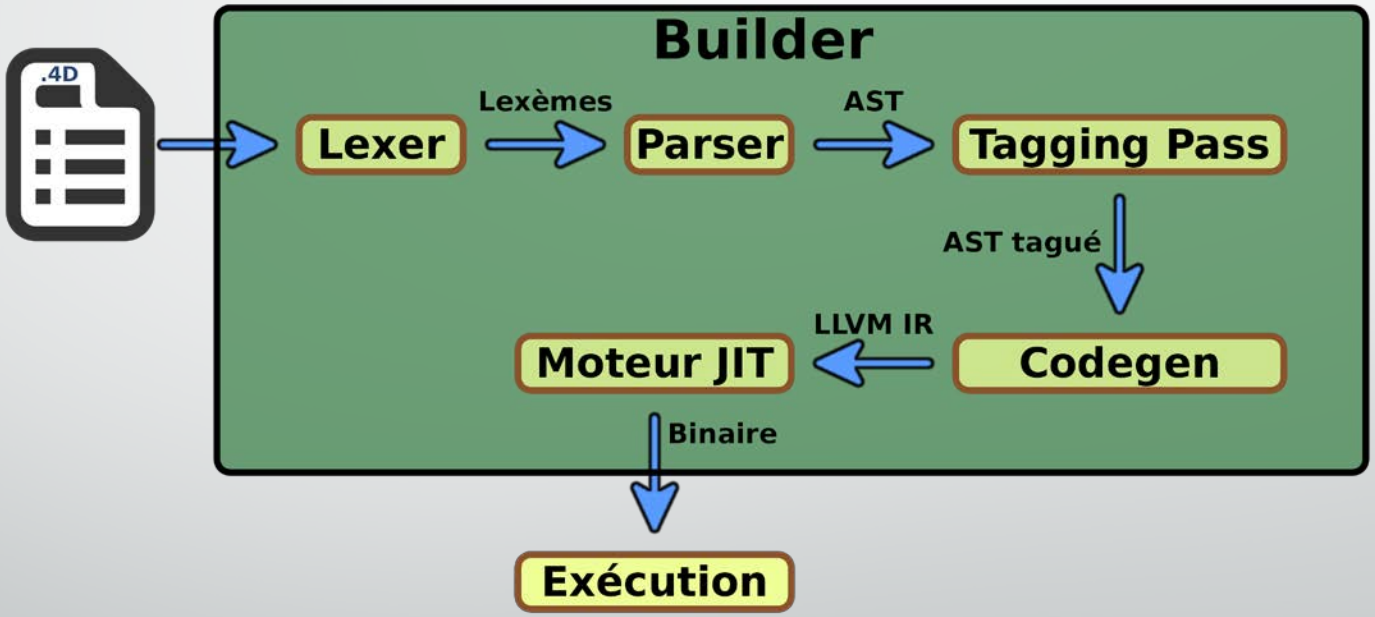
Plan

- Objectifs
- Architecture et implémentation
- Restrictions choisies
- Tests et démonstrations
- Problèmes rencontrés
- Réflexions

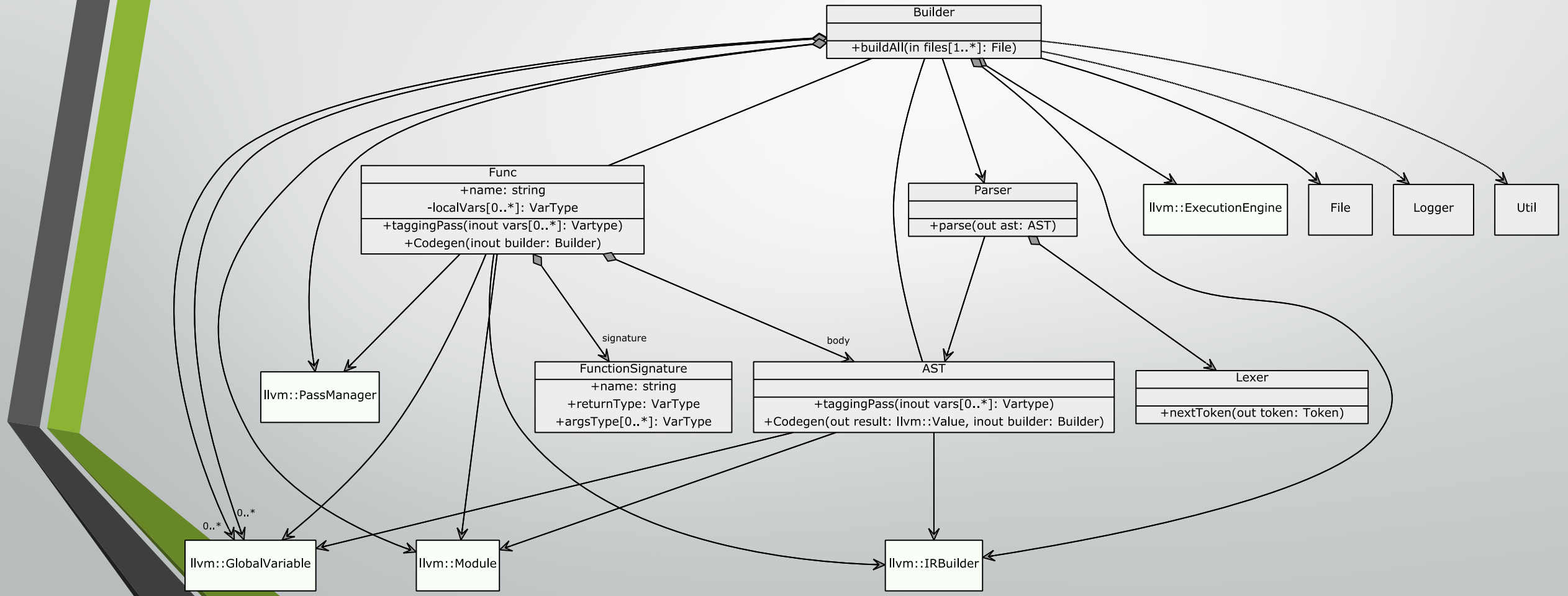
Objectifs

- Créer un compilateur 4D en utilisant la technologie LLVM
- Montrer qu'un tel compilateur est possible en en créant une version simplifiée
- Cerner les difficultés que sa réalisation présente
- Voir ses avantages
- Avoir une idée du travail nécessaire pour mettre en place un tel compilateur.

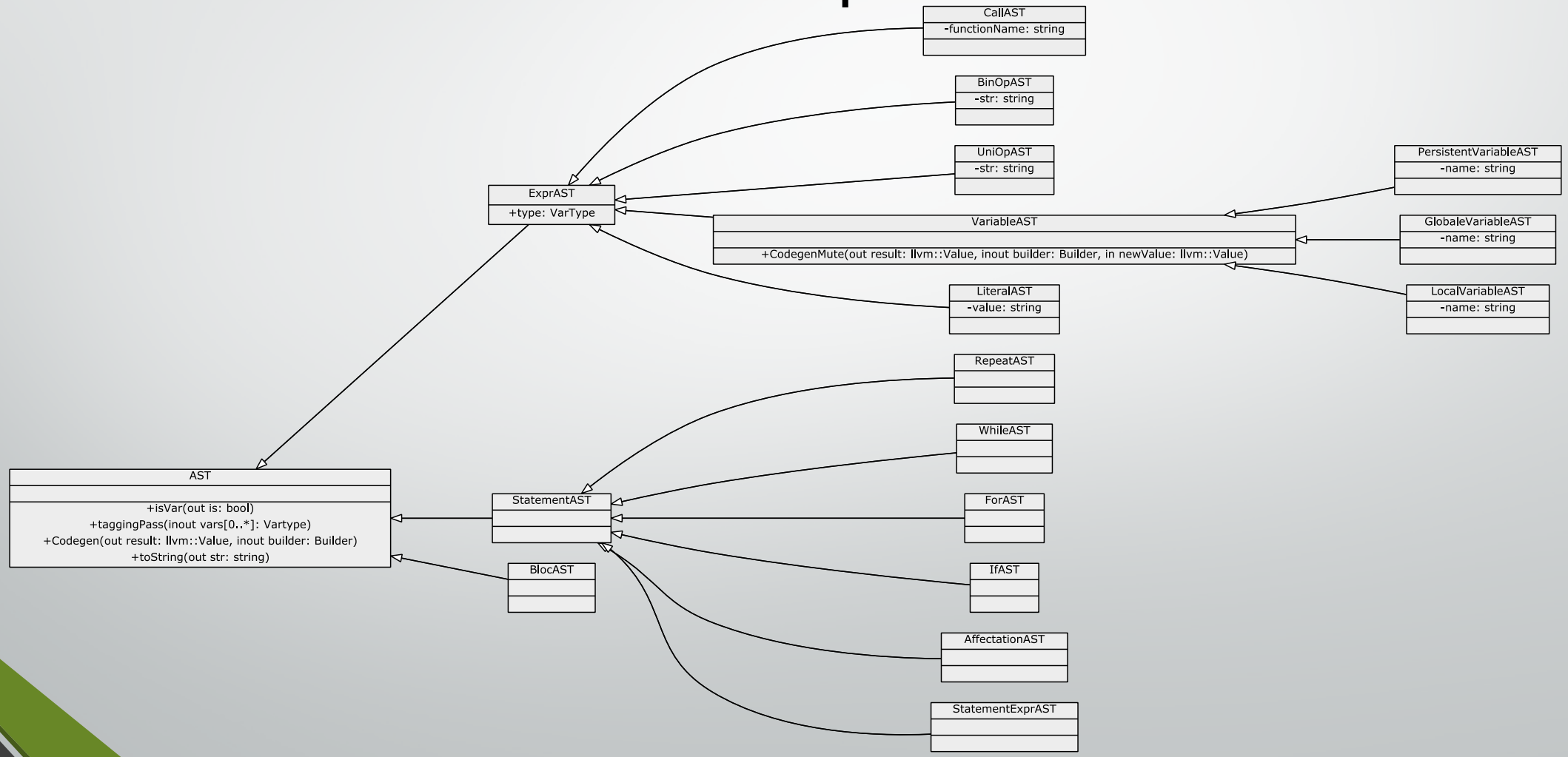
Architecture et implémentation



Architecture et implémentation



Architecture et implémentation



Architecture et implémentation

- Novembre 2014 : Création de la plateforme de développement et entraînement sur le tutoriel LLVM.
- Décembre 2014 : Création du LEXER et des premiers tests fonctionnels du LEXER
- 1ère moitié de Janvier 2015 : Création de la structure de l'AST
- 2ème moitié de Janvier 2015 : Création du PARSER (+ tests)
- Février 2015 : BUILDER et génération de code LLVM IR (+ tests). Début de parcours de l'AST en vu d'un typage des expressions.
- Mars 2015 : Fin du générateur de code, ajout des appels de fonctions (builtins ou d'autres fichiers), Refactor général.

Architecture et implémentation

Le compilateur gère les fonctionnalités suivantes :

- Les variables de type entier
- les arguments de fonctions et les retours de type entier
- Les fonctions builtins ALERT et ABORT
- les IF avec ou sans ELSE
- Les boucles WHILE et REPEAT
- Les boucles FOR avec ou sans la valeur d'incrément
- Les appels de fonctions codées dans d'autres fichiers 4D

Restrictions choisies

- Les types gérés sont uniquement les entiers
- Les pointeurs ne sont pas gérés
- Requêtes Bases de données
- Switch-case
- Objets

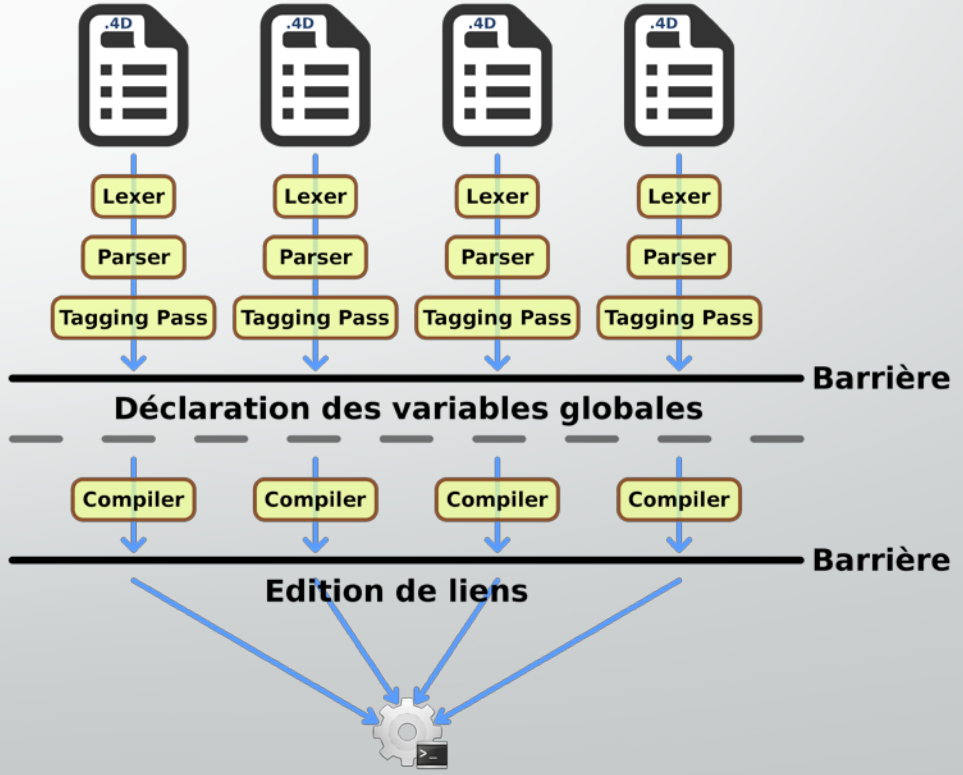
Programme de test et démonstration

Problèmes rencontrés

- Compilation d'un projet avec LLVM
- Compilation de LLVM sous Windows
- Génération de LLVM IR
- Documentation LLVM
- Limitations Imposées par LLVM

Réflexions diverses

- JIT
- Debug et LLVM
- Multi-plateforme
- Optimisation de la compilation
- Objets
- Variants



Réflexions diverses

C++

```
class Foo {
  double bar;
  int baz;
  void print() {
    printf("%lf %d", this->bar, this->baz);
  }
};

Foo* foo = new Foo();
foo->print();
```

C

```
struct Foo {
  double bar;
  int baz;
};

void _Foo_print(Foo* this) {
  printf("%lf %d", this->bar, this->baz);
}

Foo* foo = malloc(sizeof(Foo));
_Foo_print(foo);
```

Réflexions diverses

Assembleur LLVM

```
%class.Foo = type { double, i32 }

; Function Attrs: uwtable
define linkonce_odr void @_ZN3Foo5printEv(%class.Foo* %this) #2 align 2 {
  %foo.ptr.ptr = alloca %class.Foo*, align 8
  store %class.Foo* %this, %class.Foo** %foo.ptr.ptr, align 8
  %foo.ptr     = load %class.Foo** %foo.ptr.ptr
  %foo.bar.ptr = getelementptr inbounds %class.Foo* %foo.ptr, i32 0, i32 0
  %foo.bar     = load double* %foo.bar.ptr, align 8
  %foo.baz.ptr = getelementptr inbounds %class.Foo* %foo.ptr, i32 0, i32 1
  %foo.baz     = load i32* %foo.baz.ptr, align 4
  %1 = call i32 (i8*, ...)* @printf(i8* "%lf %d", double %foo.bar, i32 %foo.baz)
  ret void
}
```

Conclusion

- Compilateur fonctionnel sous Windows et Linux
- Compilation de programmes complexes sous les restrictions imposées
- Pistes pour rendre le compilateur industrialisable

Avez-vous des questions ?