Master d'informatique 2015-2016
Spécialité STL
« Implantation de langages »
DLP – 4I501
épisode ILP2

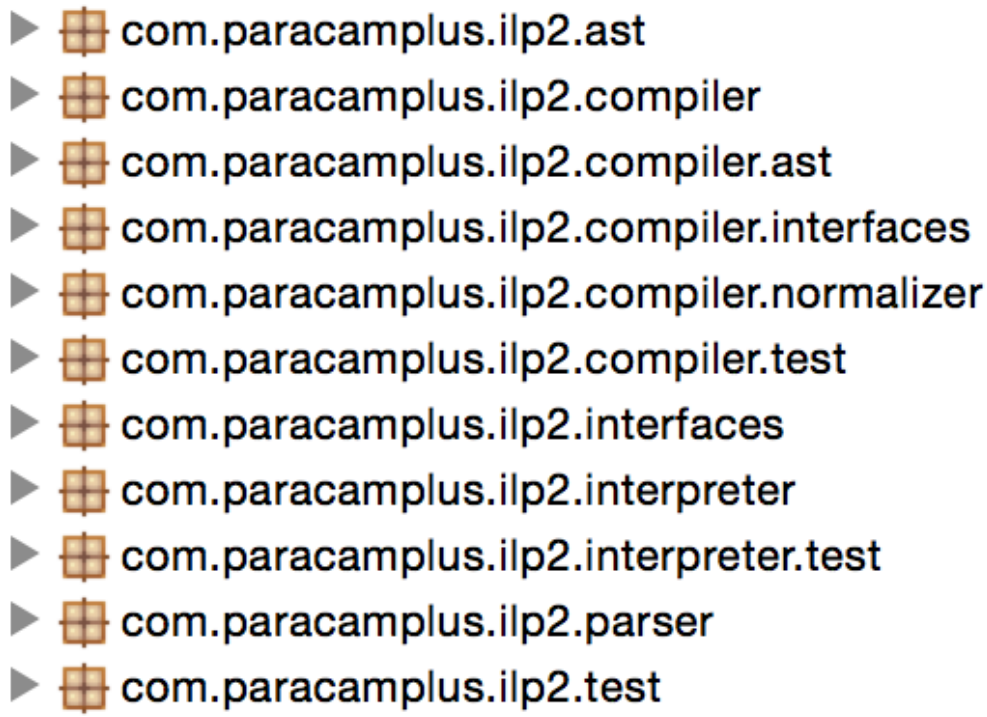
Buts

- ILP2 = ILP1 +
  - fonctions globales
  - boucle
  - affectation
- Analyse statique

Plan du cours 5

- Présentation d'ILP2
- Syntaxe
- Sémantique
- Génération de C
- Nouveautés techniques Java

Nouveaux packages

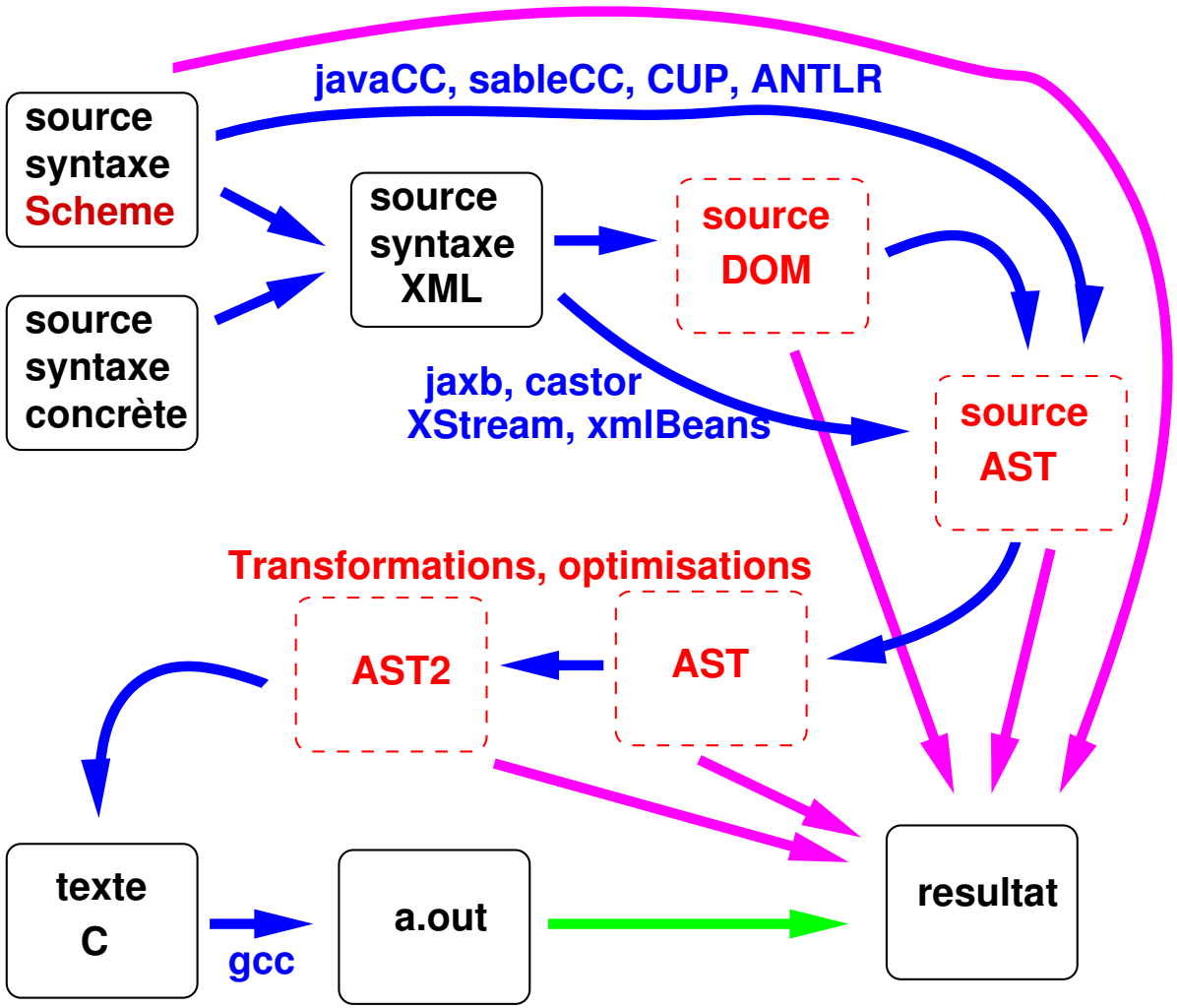


Adjonctions

ILP2 = ILP1 + définition de fonctions globales + boucle `while` + affectation.

```
let deuxfois x =
    x + x;;
let fact n = if n = 1 then 1 else n * fact (n-1);;
let x = 1 and y = "foo" in
    while x < 100 do
        x := deuxfois (fact(x));
        y := deuxfois y;
    done
    y;;
```

Grand schéma



Grammaire

```
functionDefinition = element functionDefinition {
    attribute name      { xsd:Name - ( xsd:Name { pattern = "(ilp|ILP)" } ) },
    element variables { variable * },
    element body        { expression + }
}

expression |=
   loop
 | assignment

loop = element loop {
    element condition { expression },
    element body        { expression + }
}

assignment = element assignment {
    attribute name { xsd:Name - ( xsd:Name { pattern = "(ilp|ILP)" } ) },
    element value  { expression }
}

program = element program {
    (functionDefinition) *,
    expression +
}
```

Analyseur

Une nouvelle classe *Parser* qui hérite de la classe *Parser* d'ILP1.

```
public class Parser
extends com.paracamplus.ilp1.ast.Parser {

   public Parser(IParserFactory factory) {
                super(factory);
   addMethod("assignment", Parser.class);
   addMethod("loop", Parser.class);
   addMethod("functionDefinition", Parser.class);
   }
}

public IASTexpression assignment (Element e)
        throws ParseException {
 String name = e.getAttribute("name");
 IASTexpression value = narrowToIASTexpression(
        findThenParseChildContent(e, "value"));
 IASTvariable variable = getFactory().newVariable(name);
 return
 ((IParserFactory) getFactory()).newAssignment(variable, value);
}
```

Une nouvelle fabrique

L'analyseur prend une fabrique à sa construction.

```
public class ASTfactory
extends com.paracamplus.ilp1.ast.ASTfactory  implements IParserFactory{

    public IASTprogram newProgram(IASTfunctionDefinition[] functions,
                                  IASTexpression expression) {
        return new ASTprogram(functions, expression);
    }


    public IASTassignment newAssignment(IASTvariable variable,
                                        IASTexpression value) {
        return new ASTassignment(variable, value);
    }


    public IASTloop newLoop(IASTexpression condition, IASTexpression body) {
        return new ASTloop(condition, body);
    }

    public IASTfunctionDefinition newFunctionDefinition(
            IASTvariable functionVariable,
            IASTvariable[] variables,
            IASTexpression body) {
        return new ASTfunctionDefinition(functionVariable, variables, body);
    }

}
```

Sémantique discursive

Boucle comme en C (sans sortie prématurée)
Affectation comme en C (expression) sauf que (comme en JavaScript) l'affectation sur une variable non locale crée la variable globale correspondante

```
let n = 1 in
  while n < 100 do
    f = 2 * n
  done;
print f
```

Fonctions globales en récursion mutuelle (comme en JavaScript, pas comme en C ou Pascal)

```
function pair (n) {
    if ( n == 0 ) {
        true
    } else {
        impair(n-1)
    }
}
function impair (n) {
    if ( n == 0 ) {
        false
    } else {
        pair(n-1)
    }
}
```

Nouvelles classes AST

ASTassignment, ASTloop, ASTprogram, ASTfunctionDefinition.

```
public class ASTassignment extends ASTexpression
implements IASTassignment, IASTvisitable {

    public ASTassignment (IASTvariable variable, IASTexpression expression) {
        this.variable = variable;
        this.expression = expression;
    }
    private final IASTvariable variable;
    private final IASTexpression expression;

    public IASTvariable getVariable() {
        return variable;
    }

    public IASTexpression getExpression() {
        return expression;
    }

    public <Result, Data, Anomaly extends Throwable>
    Result accept(IASTvisitor<Result, Data, Anomaly> visitor, Data data)
            throws Anomaly {
        return visitor.visit(this, data);
    }
```

Nouvelles interfaces : IASTvisitor et IASTvisitable

```
public interface
IASTvisitor<Result, Data, Anomaly extends Throwable>
extends
com.paracamplus.ilp1.interfaces.IASTvisitor
                                    <Result, Data, Anomaly>{

Result visit(IASTassignment iast, Data data)
                                    throws Anomaly;
Result visit(IASTloop iast, Data data)
                                    throws Anomaly;
}

public interface IASTvisitable {
        <Result, Data, Anomaly extends Throwable>

Result
accept(IASTvisitor<Result, Data, Anomaly> visitor,
                    Data data) throws Anomaly;
}
```

Affectation : accept

```
public <Result, Data, Anomaly extends Throwable>
Result
accept(IASTvisitor<Result, Data, Anomaly> visitor, Data data)
            throws Anomaly {
return visitor.visit(this, data);
}
```

The type ASTassignment must implement the inherited abstract method IASTvisitable.accept(IASTvisitor<Result,Data,Anomaly>, Data)

```
public <Result, Data, Anomaly extends Throwable>
Result
accept(com.paracamplus.ilp1.interfaces.IASTvisitor
                        <Result, Data, Anomaly> visitor,
                        Data data) throws Anomaly {
return ((IASTvisitor <Result, Data, Anomaly>) visitor).
                        visit(this, data);
}
```

Interprétation

```
public class Interpreter
extends com.paracamplus.ilp1.interpreter.Interpreter
implements
IASTvisitor<Object, ILexicalEnvironment, EvaluationException> {

public Interpreter(IGlobalVariableEnvironment globalVariableEnvironment,
                   IOperatorEnvironment operatorEnvironment) {
            super(globalVariableEnvironment, operatorEnvironment);
}

public Object visit(IASTprogram iast, ILexicalEnvironment lexenv)
        throws EvaluationException {
     for ( IASTfunctionDefinition fd : iast.getFunctionDefinitions() ) {
        Object f = this.visit(fd, lexenv);
        String v = fd.getName();
        getGlobalVariableEnvironment().addGlobalVariableValue(v, f);
     }
     try {
        return iast.getBody().accept(this, lexenv);
    } catch (Exception exc) {
        return exc;
     }
}
```

Interpretation : définition de fonction

```
public Invocable visit(IASTfunctionDefinition iast,
                       ILexicalEnvironment lexenv)
              throws EvaluationException {

Invocable fun =
new Function(iast.getVariables(), iast.getBody(),
                         new EmptyLexicalEnvironment());

getGlobalVariableEnvironment()
          .addGlobalVariableValue(iast.getName(), fun);

return fun;
}
```

Repose sur un nouvel objet de la bibliothèque d'exécution.

```
public class Function implements IFunction {

public Function (IASTvariable[] variables,
                 IASTexpression body,
                 ILexicalEnvironment lexenv) {
        this.variables = variables;
        this.body = body;
        this.lexenv = lexenv;
    }

 public int getArity() {
        return variables.length;
    }
...

 public Object apply(Interpreter interpreter, Object[] argument)
          throws EvaluationException {
        if ( argument.length != getArity() ) {
            String msg = "Wrong arity";
            throw new EvaluationException(msg);
        }

        ILexicalEnvironment lexenv2 = getClosedEnvironment();
        IASTvariable[] variables = getVariables();
        for ( int i=0 ; i<argument.length ; i++ ) {
            lexenv2 = lexenv2.extend(variables[i], argument[i]);
        }
        return getBody().accept(interpreter, lexenv2);
    }
}
```

Interpretation d'une boucle

```
public Object visit(IASTloop iast,
                    ILexicalEnvironment lexenv)
            throws EvaluationException {
while ( true ) {
   Object condition = iast.getCondition().
                              accept(this, lexenv);
   if ( condition instanceof Boolean ) {
        Boolean c = (Boolean) condition;
        if ( ! c ) {
                    break;
                }
            }
        iast.getBody().accept(this, lexenv);
        }
        return Boolean.FALSE;
    }
```

Interpretation d'une affectation

```
public Object visit(IASTassignment iast,
                    ILexicalEnvironment lexenv)
            throws EvaluationException {
IASTvariable variable = iast.getVariable();
Object value = iast.getExpression().accept(this, lexenv);
try {
   lexenv.update(variable, value);
} catch (EvaluationException exc) {
   getGlobalVariableEnvironment()
      .updateGlobalVariableValue(variable.getName(),
                                 value);
}
return value;
}
```

Les variables sont maintenant modifiables. Les interfaces des environnements d'interprétation doivent donc procurer cette nouvelle fonctionnalité.

Test d'interpretation

Ressource: com.paracamplus.ilp2.interpreter.test.InterpreterTest

```
public class InterpreterTest extends
com.paracamplus.ilp1.interpreter.test.InterpreterTest{

protected static String rngFileName =
        "Grammars/grammar2.rng";
protected static String samplesDirName =
        "SamplesILP2";

public InterpreterTest(final File file) {
        super (file);
        IParserFactory factory =  new ASTfactory();
        this.parser = new Parser(factory);

}

@Test
public void processFile () throws Throwable {
...
```

Compilation

```
public class Compiler
implements
IASTCvisitor<Void, Compiler.Context, CompilationException>

public Compiler (IOperatorEnvironment ioe,
                 IGlobalVariableEnvironment igve ) {
        this.operatorEnvironment = ioe;
        this.globalVariableEnvironment = igve;
    }

protected final IOperatorEnvironment
        operatorEnvironment;
protected final IGlobalVariableEnvironment
        globalVariableEnvironment;
```

Variables globales

```
;;; $Id: u59-2.scm 405 2006-09-13 17:21:53Z queinnec $
(comment "variable globale non fonctionnelle")
(let ((x 1))
  (set! g 59)
  g )

;;; end of u59-2.scm
```

Variables globales

L'affectation sur une variable non locale réclame, en C, que l'on ait déclaré au préalable cette variable globale.

1. il faut collecter les variables globales
2. pour chacune d'entre elles, il faut l'allouer et l'initialiser.

Première analyse statique : collecte des variables globales. Réalisation : par arpentage de l'AST (un visiteur).

Variables globales (suite)

```
public String compile(IASTprogram program)
            throws CompilationException {

IASTCprogram newprogram = normalize(program);
newprogram = optimizer.transform(newprogram);

GlobalVariableCollector gvc = new GlobalVariableCollector()
Set<IASTCglobalVariable> gvs = gvc.analyze(newprogram);
newprogram.setGlobalVariables(gvs);
        ...
try {
            out = new BufferedWriter(sw);
            visit(newprogram, context);
            out.flush();
} catch (IOException exc) {
      ...
```

```
/* Global variables */
ILP_Object      g;

ILP_Object
ilp_program()
{
{
ILP_Object      ilptmp209;
ilptmp209 = ILP_Integer2ILP(1);

{
        ILP_Object      x1 = ilptmp209;
        {
                ILP_Object      ilptmp210;
                {
                        ILP_Object      ilptmp211;
                        ilptmp211 = ILP_Integer2ILP(59);
                        ilptmp210 = (g = ilptmp211);
                }
                ilptmp210 = g;
                return ilptmp210;
        }

}
}

}
```

Collecte des variables globales

Toute variable non locale est globale.
Parcours récursif de la grammaire.

- GV(sequence(i1, i2, ...) = GV(i1) $\cup$ GV(i2) $\cup$ ...
- GV(alternative(c,it,if)) = GV(c) $\cup$ GV(it) $\cup$ GV(if)
- GV(boucle(c,s)) = GV(c) $\cup$ GV(s)
- GV(affectation(n,v)) = { n } $\cup$ GV(v)
- GV(constante) = $\emptyset$
- GV(variable) = { variable }
- GV(definitionFonction(n,(v1,v2,...),c) = GV(c) - { v1, v2, ... }
- GV(blocUnaire(v,e,c)) = GV(e) $\cup$ (GV(c) - { v })

Le visitor *GlobalVariableCollector*

```
public class GlobalVariableCollector
implements IASTCvisitor<Set<IASTCglobalVariable>,
                        Set<IASTCglobalVariable>,
                        CompilationException> {

public GlobalVariableCollector () {
        this.result = new HashSet<>();
}
protected Set<IASTCglobalVariable> result;

public Set<IASTCglobalVariable>
        analyze(IASTprogram program)
            throws CompilationException {
        result = program.getBody().accept(this, result);
        return result;
}
```

Le visitor *GlobalVariableCollector*

Grande partie du travail a été déjà fait par les visiteur *Normalize*

```
public Set<IASTCglobalVariable> visit(
        IASTCglobalVariable iast,
        Set<IASTCglobalVariable> result)
                throws CompilationException {
    result.add(iast);
    return result;
}

public Set<IASTCglobalVariable> visit(
        IASTClocalVariable iast,
        Set<IASTCglobalVariable> result)
                throws CompilationException {
    return result;
}


public Set<IASTCglobalVariable> visit(
        IASTalternative iast,
        Set<IASTCglobalVariable> result)
                throws CompilationException {
    result = iast.getCondition().accept(this, result);
    result = iast.getConsequence().accept(this, result);
    result = iast.getAlternant().accept(this, result);
    return result;
}
```

Il nous faut une nouvelle classe ASTCprogramme

```
public class ASTCprogram
extends
com.paracamplus.ilp1.compiler.ast.ASTCprogram
implements
com.paracamplus.ilp2.compiler.interfaces.IASTCprogram {

public ASTCprogram (IASTCfunctionDefinition[] functions,
                    IASTexpression expression) {
        super(expression);
        this.functions = Arrays.asList(functions);
}

protected List<IASTfunctionDefinition> functions;

protected Set<IASTCglobalVariable> globalVariables;

}
```

```
public Void visit(IASTCprogram iast, Context context)
        throws CompilationException {
    emit(cProgramPrefix);

    emit(cGlobalVariablesPrefix);
    for ( IASTCglobalVariable gv : iast.getGlobalVariables() ) {
        emit("ILP_Object ");
        emit(gv.getMangledName());
        emit(";\n");
    }

    emit(cPrototypesPrefix);
    Context c = context.redirect(NoDestination.NO_DESTINATION);
    for ( IASTfunctionDefinition ifd : iast.getFunctionDefinitions() ) {
        this.emitPrototype(ifd, c);
    }

    emit(cFunctionsPrefix);
    for ( IASTfunctionDefinition ifd : iast.getFunctionDefinitions() ) {
        this.visit(ifd, c);
        emitClosure(ifd, c);
    }

    emit(cBodyPrefix);
    Context cr = context.redirect(ReturnDestination.RETURN_DESTINATION);
    iast.getBody().accept(this, cr);
    return null;
}
```

Fonctions : compilation

fonctionGlobale = (nom, variables..., corps)

$\overrightarrow{fonctionGlobale}$

```
// Declaration
static ILP_Object nom (
        ILP_Object variable, ... );
...
// Definition
ILP_Object nom (
        ILP_Object variable,
        ...
) {
    →return
      corps
}
```

Compilation des fonctions

Pour le prototype

```
protected void emitPrototype(IASTCfunctionDefinition iast, Context context)
        throws CompilationException {
    emit("ILP_Object "); emit(iast.getCName());emit("(\n");
    IASTvariable[] variables = iast.getVariables();
    for ( int i=0 ; i< variables.length ; i++ ) {
        IASTvariable variable = variables[i];
        emit(",\n    ILP_Object ");
        emit(variable.getMangledName());
    }
    emit(");\n");
}
```

Pour la définition

```
public Void visit(IASTCfunctionDefinition iast, Context context)
        throws CompilationException {

    // Idem que pour le prototype
    emit(") {\n");
    Context c = context.redirect(ReturnDestination.RETURN_DESTINATION);
    iast.getBody().accept(this, c);
    emit("}\n");
    return null;
}
```

Boucle : compilation

Il y a un équivalent en C que l'on emploie !
boucle = (condition, corps)

$\longrightarrow$d
boucle

```
while ( ILP_isEquivalentToTrue( condition ) ) {
   →(void)
     corps    ;
}
     →d
nImporteQuoi  ;
```

Compilation de la boucle

L'implantation :

```
public Void visit(IASTloop iast, Context context)
        throws CompilationException {
 emit("while ( 1 ) { \n");
 IASTvariable tmp = context.newTemporaryVariable();
 emit("  ILP_Object " + tmp.getMangledName() + "; \n");
 Context c = context.redirect(new AssignDestination(tmp));
 iast.getCondition().accept(this, c);
 emit("  if ( ILP_isEquivalentToTrue(");
 emit(tmp.getMangledName());
 emit(") ) {\n");
 Context cb = context.redirect(VoidDestination.VOID_DESTINATION);
 iast.getBody().accept(this, cb);
 emit("\n} else { \n");
 emit("    break; \n");
 emit("\n}\n}\n");
 whatever.accept(this, context);
 return null;
 }
```

Boucle : exemple

```
;;; $Id: u52-2.scm 405 2006-09-13 17:21:53Z queinnec $
(comment "boucle tant-que")
(let ((x 50))
  (while (< x 52)
    (set! x (+ x 1)) )
  x )

;;; end of u52-2.scm
```

```
ILP_Object      ilptmp141;
ilptmp141 = ILP_Integer2ILP(50);
{
        ILP_Object      x1 = ilptmp141;
        {
        ILP_Object      ilptmp142;
                while (1) {
                ILP_Object      ilptmp143;
                {
                        ILP_Object      ilptmp144;
                        ILP_Object      ilptmp145;
                        ilptmp144 = x1;
                        ilptmp145 = ILP_Integer2ILP(52);
                        ilptmp143 = ILP_LessThan(ilptmp144, ilptmp145);
                }
                if (ILP_isEquivalentToTrue(ilptmp143)) {
                {
                        ILP_Object      ilptmp146;
                        {
                                ILP_Object      ilptmp147;
                                ILP_Object      ilptmp148;
                                ilptmp147 = x1;
                                ilptmp148 = ILP_Integer2ILP(1);
                                ilptmp146 = ILP_Plus(ilptmp147, ilptmp148);
                        }
                        (void)(x1 = ilptmp146);
                        }

                } else {
                        break;

                }
        }
        ilptmp142 = ILP_FALSE;
        ilptmp142 = x1;
        return ilptmp142;
        }

}
```

Affectation : compilation

Là encore, on utilise les ressources de C.
affectation = (variable, valeur)

$$\xrightarrow{}\texttt{d}\ \textit{affectation}$$

$$\texttt{d}\ (\ \xrightarrow{}\texttt{variable}\ \textit{valeur}\ )$$

Affectation : génération de code

```
private Void visitNonLocalAssignment
        (IASTassignment iast, Context context)
        throws CompilationException {
IASTvariable tmp1 = context.newTemporaryVariable();
emit("{ \n");
emit("  ILP_Object " + tmp1.getMangledName() + "; \n");
Context c1 = context.redirect(new AssignDestination(tmp1))
iast.getExpression().accept(this, c1);
emit(context.destination.compile());
emit("(");
emit(iast.getVariable().getMangledName());
emit(" = ");
emit(tmp1.getMangledName());
emit("); \n} \n");
return null;
}
```

Test de compilation

Ressource: com.paracamplus.ilp2.compiler.test.CompilerTest

```
@RunWith(Parameterized.class)
public class CompilerTest {

    protected static String rngFileName = "Grammars/grammar2.rng";
    protected static String samplesDirName = "SamplesILP2";
    protected static String pattern = "ur?[0-78]\\d*-[123456](gfv)?";

    public CompilerTest(final File file) {
        this.file = file;
        IParserFactory factory = new com.paracamplus.ilp2.ast.ASTfactory();
        this.parser = new com.paracamplus.ilp2.ast.Parser(factory);
    }
    protected File file;

    public void setParser (IParser parser) {
        this.parser = parser;
    }
    protected IParser parser;


    @Test
    public void processFile () throws Throwable {
```

Heritage des visiteurs

```
public interface
IASTCvisitor<Result, Data, Anomaly extends Throwable>
extends com.paracamplus.ilp2.interfaces.IASTvisitor
                                        <Result, Data, Anomaly>
 {

Result visit(IASTCglobalFunctionVariable iast, Data data
                  throws Anomaly;
Result visit(IASTCglobalInvocation iast, Data data)
                  throws Anomaly;

}
```

Un compromis : le visiteurs

```
public interface IASTvisitor
<Result, Data, Anomaly extends Throwable> {
    Result visit(IASTalternative iast, Data data)
        throws Anomaly;
    Result visit(IASTbinaryOperation iast, Data data)
         throws Anomaly;
    Result visit(IASTblock iast, Data data)
        throws Anomaly;
...
    Result visit(IASTinvocation iast, Data data)
         throws Anomaly;
    Result visit(IASToperator iast, Data data)
         throws Anomaly;
}

public interface IASTvisitable {
    <Result, Data, Anomaly extends Throwable>
    Result accept(IASTvisitor<Result, Data, Anomaly> visitor,
                  Data data) throws Anomaly;
}
```

Raffinement, spécialisation (*override*)

```
public class C {
    public void crunch (C c) {
        // utiliser this et c
    }
}
public class Sc extends C {
    @Overriding
    public void crunch (C c) {
        // utiliser this, c et super.crunch()
    }
}
```

Surcharge (*overload*)

Facilité d'écriture !

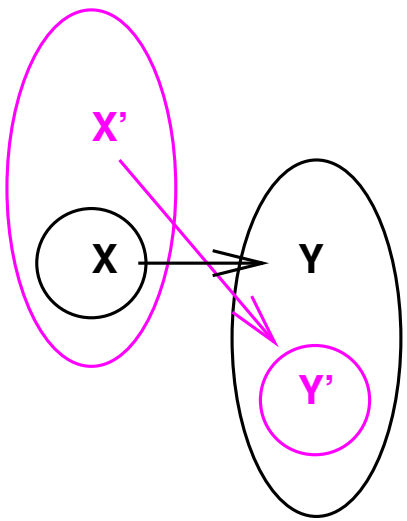
```
public class C {
   public Truc crunch(Integer i) {..}
   public Chose crunch(String d) {..}
   public Muche crunch(Object d) {..}
   void utilisation (Object o) {
      crunch(3); // boxing automatique
      if ( o instanceof Integer ) {
        crunch(o);
}
```

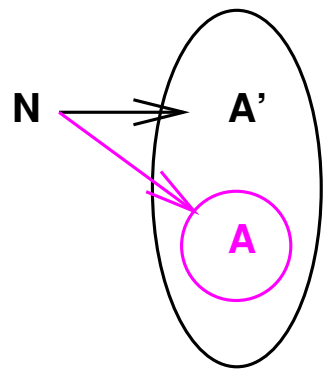
Peut se réécrire **statiquement** en :

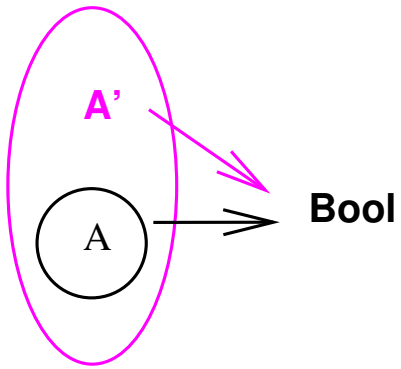
```
public class C {
   public Truc crunch_integer(Integer i) {..}
   public Chose crunch_double(String d) {..}
   public Muche crunch_object(Object d) {..}
   void utilisation (Object o) {
      crunch_integer(new Integer(3));
      if ( o instanceof Integer ) {
        crunch(o); // != crunch_integer(o)
                   // = crunch_object(o)
}
```

Contravariance

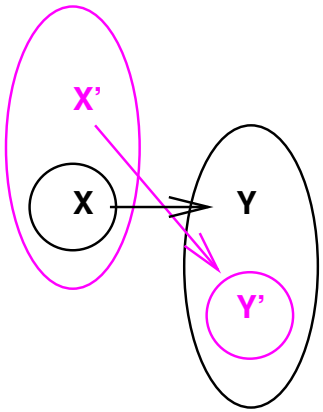
A est un sous-type de B si un $a \in A$ peut remplacer un $b \in B$ dans tous ses emplois possibles.
Une fonction $X' \to Y'$ est un sous-type de $X \to Y$ ssi $X \subset X'$ et $Y' \subset Y$.

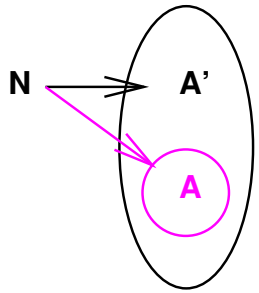


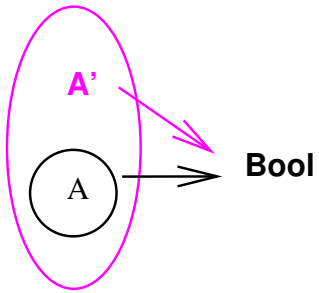




NB : J'utilise l'inclusion ensembliste comme notation pour le sous-typage.
N est l'ensemble des entiers.







Cas des tableaux : si $A \subset A'$ alors $N \to A$ sous-type de $N \to A'$ donc `A[]` sous-type de `A'[]`.
Attention, en Java, le type d'un tableau est statique et ne dépend pas du type réel de ses éléments.

A est un sous-type de B si un $a \in A$ peut remplacer un $b \in B$ **dans tous ses emplois possibles**.

Emplois possibles : A, A.length, instanceof A[], A[i] alors

```
PointColore[] pcs = new PointColore[]{ new PointColore(
assert pcs[0] instanceof Point; // OK
ps[0] = new PointColore();      // OK
Point[] ps = (Point[]) pcs;     // OK
```

Mais pour un tableau, en fait, les emplois sont A, A.length, instanceof A[], A[i] et A[i] = v

```
Point[] ps = new Point[]{ new PointColore() };
assert ps[0] instanceof Point; // OK
ps[0] = new PointColore();     // OK

// PointColore[] pcs = (PointColore[]) ps;   // FAUX!
PointColore[] pcs = new PointColore[ps.length];
for ( int i=0 ; i<pcs.length ; i++ ) {
   pcs[i] = (PointColore) ps[i];
}
ps = (Point[]) pcs;                 // OK
ps[0] = new Point(); // ArrayStoreException
```

Génériques

http://java.sun.com/j2se/1.5/pdf/generics-tutorial.pdf

```
public interface List<E, T extends Exception> {
  void add(E x);
  Iterator<E> iterator();
  E getOne() throws T;
}
```

Généricité

$\forall$ E sous-type de Exception, alors la méthode frobnicate doit prendre une liste de E et rendre une liste de E :

```
public interface Igeneric<E extends Exception> {
    List<E> frobnicate (List<E> es) throws E;
    <T extends E> Collection<T> crunch(List<E> es);
}
```

$\forall$ T sous-type de E, la méthode crunch doit prendre une liste de E et rendre une collection de T.

Généricité suite

La généricité en Java est implantée à l'exécution par effacement de types. La question `o instanceof List<Point>` n'a donc pas de sens. On peut cependant écrire `o instanceof List<?>`
Il n'est pas non plus possible d'écrire :

```
public void crunch(Set<Integer> si) { ...}
public void crunch(Set<String> ss) { ...}
```