AST

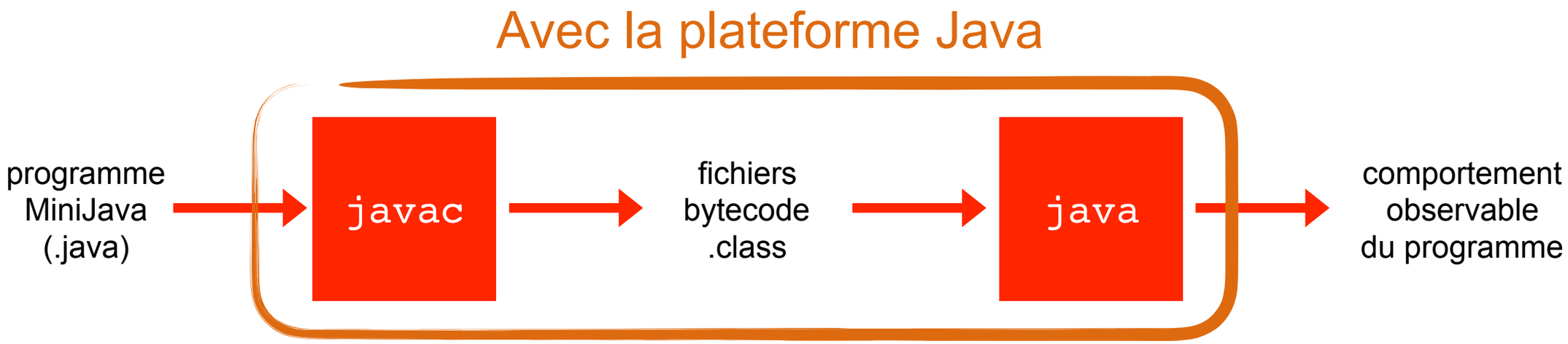
Analyse statique pour l'optimisation de programme

16 janvier 2020

David Pichardie

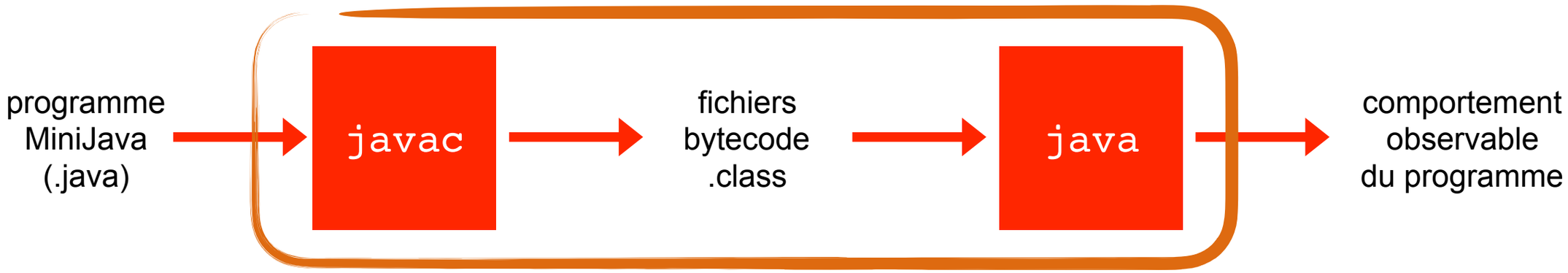
Architecture globale du projet AST

Architecture globale du projet AST



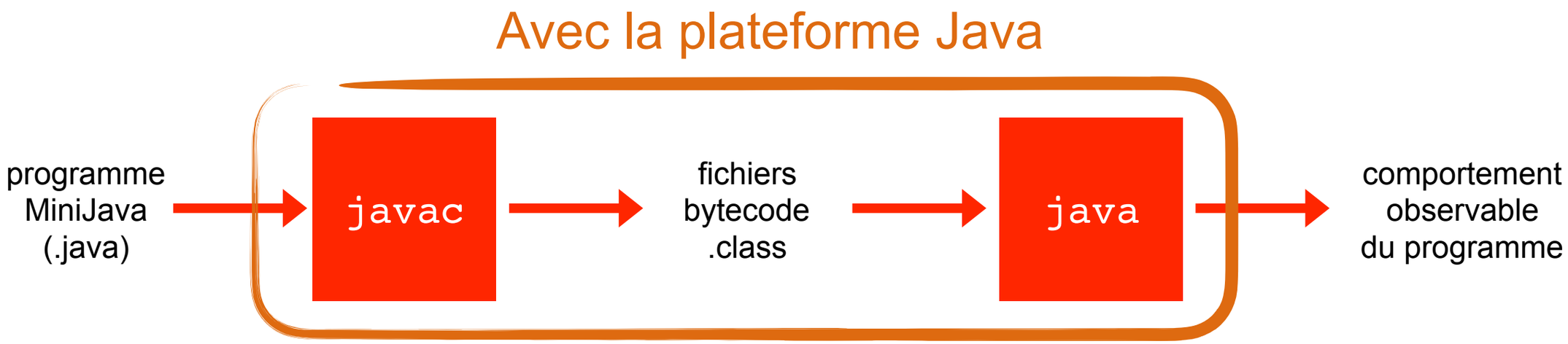
Architecture globale du projet AST

Avec la plateforme Java



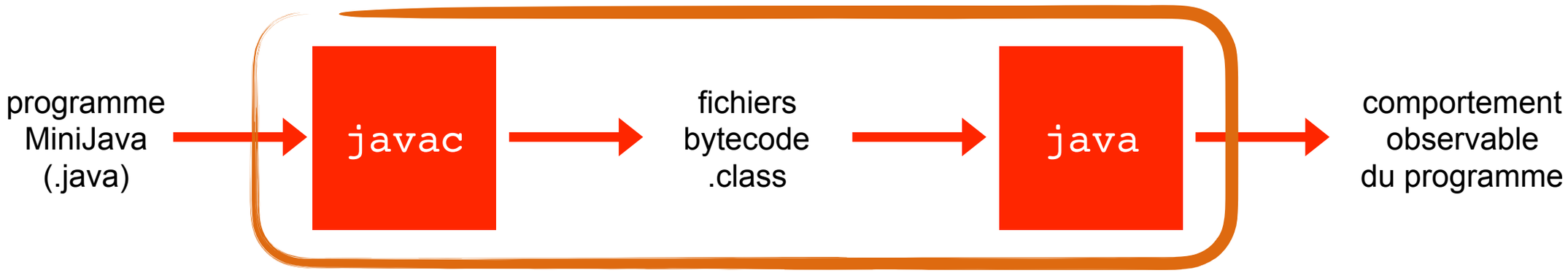
Interprétation directe

Architecture globale du projet AST

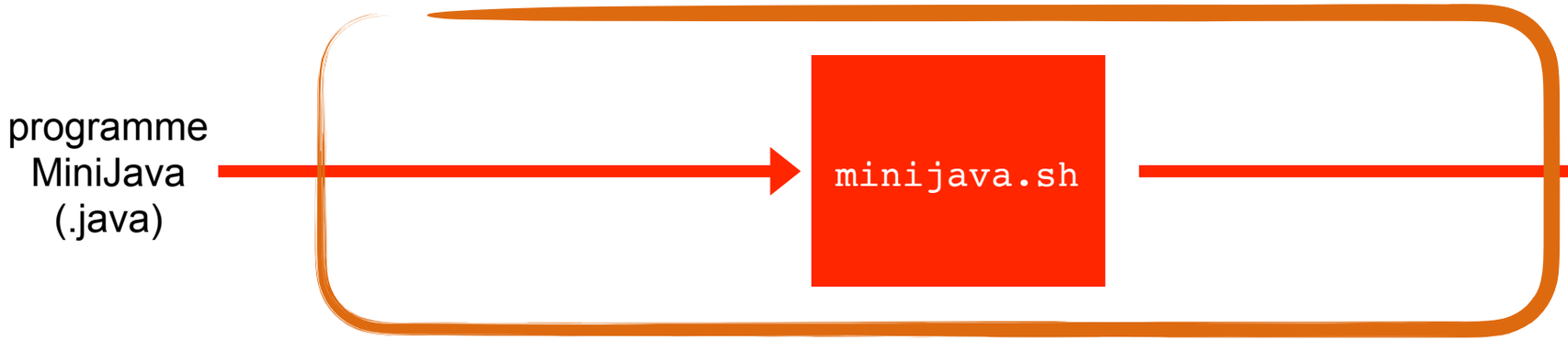


Architecture globale du projet AST

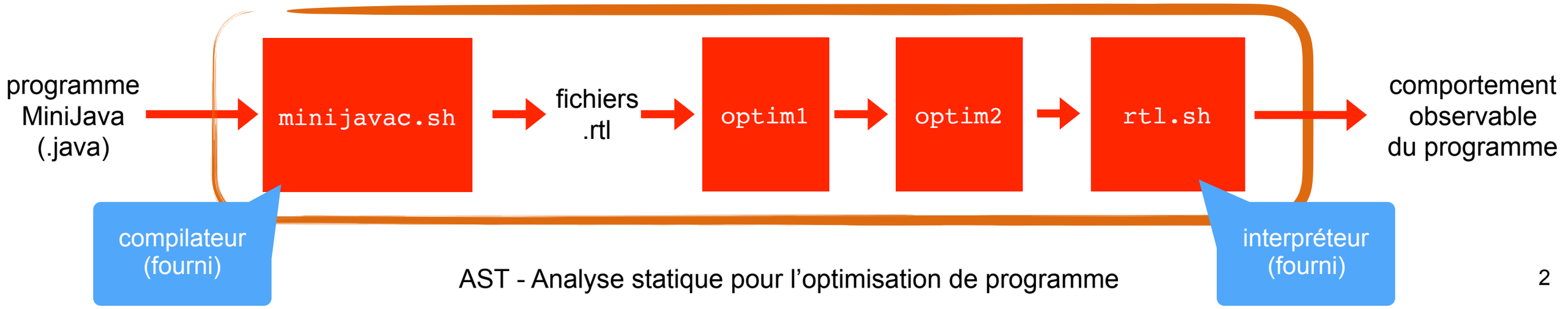
Avec la plateforme Java



Interprétation directe



Compilation, optimisation et interprétation



Mini-Java : syntaxe

$$
\begin{array}{rrcl}
(Goal) & g & ::= & mc\ d_1\ \ldots\ d_n \\
(MainClass) & mc & ::= & \texttt{class}\ id\ \{\ \texttt{public static void main (String [] }\ id^S\texttt{)\{} \\
 & & & \quad t_1\ id_1;\ \ldots;\ t_r\ id_r;\ s_1 \ldots\ s_q\}\} \\
(TypeDeclaration) & d & ::= & \texttt{class}\ id\ \{\ t_1\ id_1;\ \ldots;\ t_f\ id_f;\ m_1\ \ldots\ m_k\ \} \\
 & & | & \texttt{class}\ id\ \texttt{extends}\ id^P\ \{\ t_1\ id_1;\ \ldots;\ t_f\ id_f;\ m_1\ \ldots\ m_k\ \} \\
(MethodDeclaration) & m & ::= & \texttt{public}\ t\ id^M\ (t_1^F\ id_1^F,\ \ldots,\ t_n^F\ id_n^F)\ \{ \\
 & & & \quad t_1\ id_1;\ \ldots;\ t_r\ id_r;\ s_1\ \ldots\ s_q\ \texttt{return}\ e;\ \} \\
(Type) & t & ::= & \texttt{int[]} \mid \texttt{boolean} \mid \texttt{int} \mid id \\
(Statement) & s & ::= & \{\ s_1\ \ldots\ s_q\ \} \mid id\ \texttt{=}\ e; \mid id\ \texttt{[}\ e_1\ \texttt{]}\ \texttt{=}\ e_2; \\
 & & | & \texttt{if (}\ e\ \texttt{)}\ s_1\ \texttt{else}\ s_2 \mid \texttt{while (}\ e\ \texttt{)}\ s \mid \texttt{System.out.println(}\ e\ \texttt{);} \\
(Expression) & e & ::= & p_1\ \texttt{\&\&}\ p_2 \mid p_1\ \texttt{<}\ p_2 \mid p_1\ \texttt{+}\ p_2 \mid p_1\ \texttt{-}\ p_2 \mid p_1\ \texttt{*}\ p_2 \mid p_1\ \texttt{[}\ p_2\ \texttt{]} \\
 & & | & p\ \texttt{.length} \mid p\ .id\ (e_1,\ \ldots,\ e_n) \mid p \\
(PrimaryExpression) & p & ::= & c \mid \texttt{true} \mid \texttt{false} \mid id \mid \texttt{this} \mid \texttt{new int[}e\texttt{]} \mid \texttt{new}\ id\texttt{()} \mid \texttt{!}e \mid \texttt{(}e\texttt{)} \\
(IntegerLiteral) & c & ::= & \langle \text{INTEGER\_LITERAL} \rangle \\
(Identifier) & id & ::= & \langle \text{IDENTIFIER} \rangle
\end{array}
$$

pas d'héritage

Exercice

- Écrire un programme MiniJava contenant une classe Point.
- Chaque instance de la classe Point aura une abscisse et une ordonnée entière.
- Une méthode d'instance equals() permettra de tester l'égalité entre points.
- La procédure principale testera tout cela sur un petit exemple.

```
class MPoint {
  public static void main(String[] a) {
    Point p1;
    Point p2;
    Point p3;
    int call;

    p1 = new Point(); p2 = new Point(); p3 = new Point();
    call = p1.setX(1); call = p1.setY(1);
    call = p2.setX(1); call = p2.setY(2);
    call = p3.setX(1); call = p3.setY(1);

    if (! p1.equals(p2)) {
      System.out.println(1);
    } else {
      System.out.println(0);
    }
    if (p1.equals(p3)) {
      System.out.println(2);
    } else {
      System.out.println(0);
    }
  }
}
```

```
class Point {

  int x;
  int y;

  public int setX(int nx) {
    x = nx;
    return 0;
  }

  public int setY(int ny) {
    y = ny;
    return 0;
  }

  public int getX() {
    return x;
  }

  public int getY() {
    return y;
  }

  public boolean equals(Point that) {
    return ((((x < ((that.getX()) + 1)) &&
            ((that.getX()) < (x + 1))) &&
            (y < ((that.getY()) + 1))) &&
            ((that.getY()) < (y + 1)) );
  }
}
```

(x == that.getX()) && (y == that.getY())

Le langage RTL

- Inspiré des représentations intermédiaires des compilateurs (LLVM, GCC)
- … mais simplifié pour des raisons pédagogiques

Exemple

```
class Factorial{
  public static void main(String[] a){
    System.out.println(new Fac().ComputeFac(10));
  }
}

class Fac {
  public int ComputeFac(int num){
    int num_aux ;
    if (num < 1)
      num_aux = 1 ;
    else
      num_aux = num * (this.ComputeFac(num-1));
    return num_aux ;
  }
}
```

minijavac.sh

```
func Main(a)
  entry:
    t.1 = Alloc(1)
    t.0 = call Fac.ComputeFac(t.1 10)
    PrintInt(t.0)
    ret

func Fac.ComputeFac(this num)
  entry:
    t.0 = Lt(num 1)
    if t.0 goto if0_then else if0_else
if0_then:
    num_aux = 1
    goto if0_end
if0_else:
    t.2 = Sub(num 1)
    t.1 = call Fac.ComputeFac(this t.2)
    num_aux = Mul(num t.1)
    goto if0_end
if0_end:
    ret num_aux
```

RTL

Avec toujours une fonction *Main*

- Un programme est une liste de fonctions
- Chaque fonction contient
  - un nom
  - une liste de paramètres
  - une liste de blocs

Avec toujours un bloc *entry*

- Chaque bloc contient
  - un label
  - une liste d'instructions
  - une instruction de sortie

```
func Main(a)
  entry:
    t.1 = Alloc(1)
    t.0 = call Fac.ComputeFac(t.1 10)
    PrintInt(t.0)
    ret

func Fac.ComputeFac(this num)
  entry:
    t.0 = Lt(num 1)
    if t.0 goto if0_then else if0_else
if0_then:
    num_aux = 1
    goto if0_end
if0_else:
    t.2 = Sub(num 1)
    t.1 = call Fac.ComputeFac(this t.2)
    num_aux = Mul(num t.1)
    goto if0_end
if0_end:
    ret num_aux
```

Instructions

(Instr) *i* ::=	*id* **=** *op*	(Assign)
|	**PrintInt**(*op*)	(BuiltIn)
|	*id* **= Alloc**(*op*$_1$)	(BuiltIn)
|	*id* **= Add**(*op*$_1$ *op*$_2$)	(BuiltIn)
|	*id* **= Sub**(*op*$_1$ *op*$_2$)	(BuiltIn)
|	*id* **= Mul**(*op*$_1$ *op*$_2$)	(BuiltIn)
|	*id* **= Lt**(*op*$_1$ *op*$_2$)	(BuiltIn)
|	*id* **= And**(*op*$_1$ *op*$_2$)	(BuiltIn)
|	*id* **= call** *F*(*op*$_1$ … *op*$_n$)	(Call)
|	*id*$_2$ **= [** *id*$_1$ **+/-** *i* **]**	*(MemRead)*
|	**[** *id* **+/-** *i* **]** = *op*	*(MemWrite)*

Alloue un bloc mémoire de taille *op*$_1$, initialisé avec des zéros, et renvoie son adresse

lit/écrit à l'adresse contenue dans *id*$_1$, plus ou moins l'entier i

(EndInstr) *ei* ::=	**return** *op*	(Return)
|	**return**	(Return)
|	**goto** *label*	(Goto)
|	**if** *op* **goto** *label*$_1$ **else** *label*$_2$	(Branch)

(Operand) *op* ::=	*id*	(Ident)
|	*i*	(LitInt)

Exercice

- Écrire un programme RTL représentant la version compilée du programme MiniJava suivant

```
class Simple {
  public static void main(String[] a) {
  }
}
```

```
class T {

  int s;
  int[] t;

  public int init(int size) {
    s = size;
    t = new int[size];
    return 0;
  }

  public int size() {
    return t.length;
  }

}
```

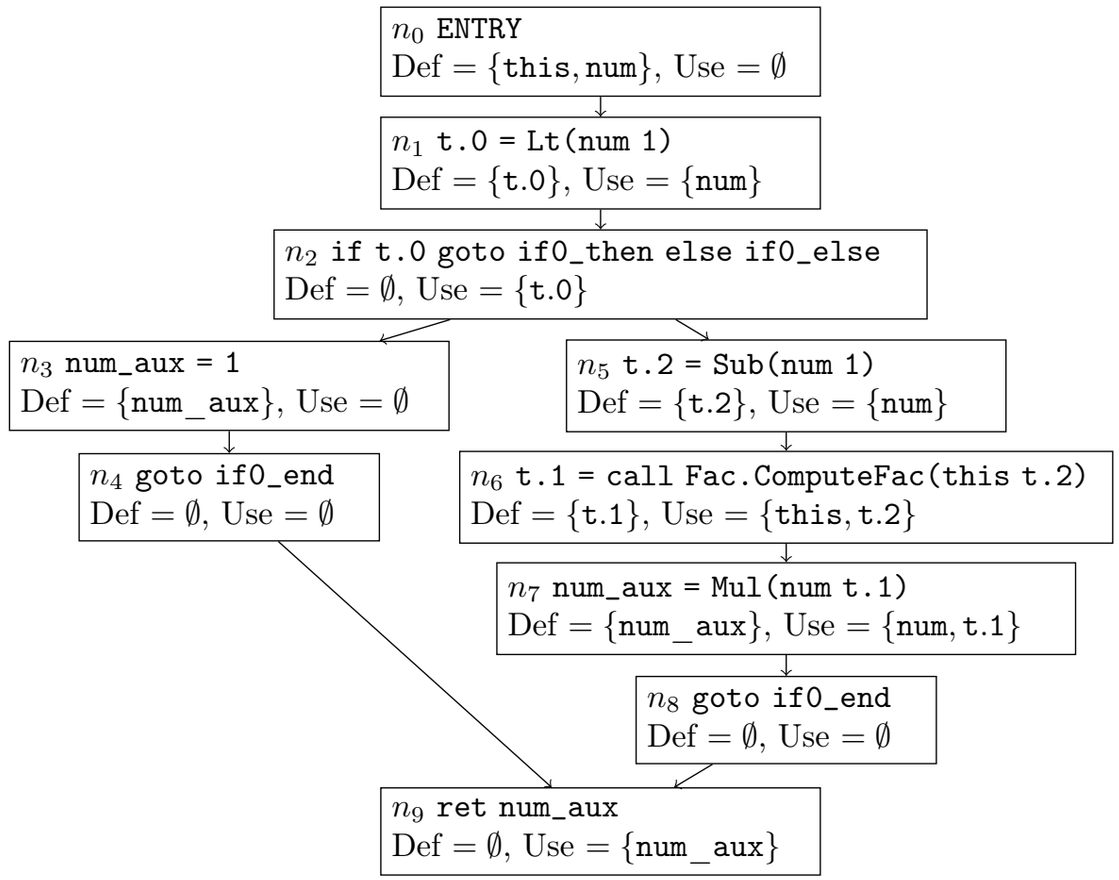
```
func Main(a)
  entry:
    x = Alloc(2)
    y = call T.init(x 10)
    t.0 = call T.size(x)
    PrintInt(t.0)
    ret

func T.init(this size)
  entry:
    [this+0] = size
    t.1 = Add(1 size)
    t.0 = Alloc(t.1)
    [t.0+0] = size
    [this+1] = t.0
    ret 0

func T.size(this)
  entry:
    t.1 = [this+1]
    t.0 = [t.1+0]
    ret t.0
```

Graphe de flot de contrôle

- On souhaite associer à chaque fonction RTL, un *graphe de flot de contrôle*
  - les noeuds correspondent à une instruction (normale ou terminale), ou bien au noeud special d'entrée
  - les arcs relient chaque paire d'instructions consécutives dans l'exécution d'une fonction
  - chaque noeud contient l'ensemble des variables utilisées (*use*) et définies (*def*) par chaque instruction
  - le noeud d'entrée ne contient que des définitions : la liste des paramètres de la fonction

Graphe de flot de contrôle

```
func Fac.ComputeFac(this num)
  entry:
    t.0 = Lt(num 1)
    if t.0 goto if0_then else if0_else
  if0_then:
    num_aux = 1
    goto if0_end
  if0_else:
    t.2 = Sub(num 1)
    t.1 = call Fac.ComputeFac(this t.2)
    num_aux = Mul(num t.1)
    goto if0_end
  if0_end:
    ret num_aux
```



Graphe de flot de contrôle

Exercice : construire le CFG de la fonction suivante

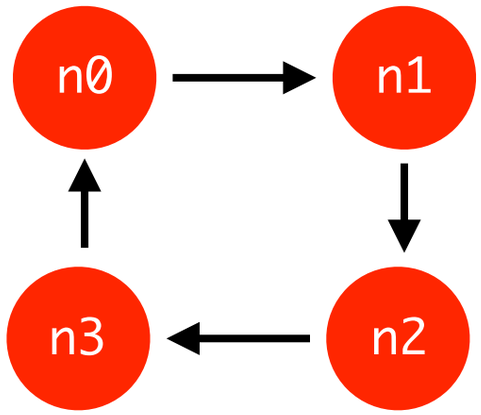
```
func T.run(this size)
  entry:
    t.1 = Add(1 size)
    t.0 = Alloc(t.1)
    [t.0] = size
    [this+1] = t.0
    goto while0_test
  while0_test:
    t.2 = Lt(i size)
    if t.2 goto while0_body else while0_end
while0_body:
  t.4 = Sub(size i)
  t.3 = [this+1]
  t.5 = Add(t.3 i)
  [t.5+1] = t.4
  i = Add(i 1)
  goto while0_test
while0_end:
  t.7 = [this+1]
  t.8 = Add(t.7 O)
  t.6 = [t.8+1]
  PrintInt(t.6)
  ret 0
```

Implémentation Java

Graphe orienté

```
public class DiGraph {
    public Set<Node> nodes();
    public void addEdge(Node from, Node to);
    public void rmEdge(Node from, Node to);
    public void show(java.io.PrintStream out);

    public class Node {
        public Node();
        public Set<Node> succ();
        public Set<Node> pred();
        public int inDegree();
        public int outDegree();
        public boolean goesTo(Node n);
        public boolean comesFrom(Node n);
    }
}
```

Exemple d'utilisation




```
import graph.DiGraph;
import graph.DiGraph.Node;

DiGraph g = new DiGraph();
Node n1 = g.new Node();
Node n2 = g.new Node();
Node n3 = g.new Node();
Node n4 = g.new Node();

g.addEdge(n1,n2);
g.addEdge(n2,n3);
g.addEdge(n3,n4);
g.addEdge(n4,n1);

g.show(System.out);
```

affiche

```
n0: n1
n1: n2
n2: n3
n3: n0
```

Control Flow Graph

```
public abstract class FlowGraph extends DiGraph {

    public abstract Set<Ident> def(Node node);

    public abstract Set<Ident> use(Node node);

    public abstract Node entry();

    public void show(java.io.PrintStream out);

}
```

Les collections Java

- List, Set et Map
  - Ne stocke que des objets (pas de types primitifs)
  - Des structures mutables : pensez à faire des copies !

Les collections Java

```
interface Set<E> implements Collection<E>, Iterable<E> {
  boolean add(E e)
  //Adds the specified element to this set if it is not already present.

  boolean addAll(Collection<? extends E> c)
  //Adds all of the elements in the specified collection to this set if
  //they're not already present.

  void clear()
  //Removes all of the elements from this set.

  boolean remove(Object o)
  //Removes the specified element from this set if it is present.

  boolean removeAll(Collection<?> c)
  //Removes from this set all of its elements that are contained in
  //the specified collection.
}

class HashSet<E> implements Set<E> {
  HashSet()
  //Constructs a new, empty set.

  HashSet(Collection<? extends E> c)
  //Constructs a new set containing the elements in the specified collection.
}

interface Map<K,V> {
  V get(Object key)
  //Returns the value to which the specified key is mapped, or null if this map
  //contains no mapping for the key.

  V put(K key, V value)
  //Associates the specified value with the specified key in this map.
}
```

Arbres de syntaxe abstraite en Java : *visitor pattern*

- Ces classes implémentent une syntaxe abstraite pour les expressions arithmétiques
- On veut programmer des *services* sur ces expressions, ici un interpréteur
- Mais on voudrait éviter d'écrire un bout de cet interpréteur dans chaque classe.

```
public abstract class Exp {
    public abstract int eval(Map<String,Integer> env);
}
public class PlusExp extends Exp {
    private Exp e1,e2;
    public PlusExp(Exp a1, Exp a2) { e1=a1; e2=a2; }
    public int eval(Map<String,Integer> env) {
        return e1.eval(env)+e2.eval(env);
    }
}
public class MinusExp extends Exp {
    private Exp e1,e2;
    public MinusExp(Exp a1, Exp a2) { e1=a1; e2=a2; }
    public int eval(Map<String,Integer> env) {
        return e1.eval(env)-e2.eval(env);
    }
}
public class TimesExp extends Exp {
    private Exp e1,e2;
    public TimesExp(Exp a1, Exp a2) { e1=a1; e2=a2; }
    public int eval(Map<String,Integer> env) {
        return e1.eval(env)*e2.eval(env);
    }
}
public class IntegerLiteral extends Exp {
    private int val;
    public IntegerLiteral(int i) { val = i; }
    public int eval(Map<String,Integer> env) {
        return val;
    }
}
public class Identifier extends Exp {
    private String s;
    public Identifier(String n) { s = n; }
    public int eval(Map<String,Integer> env) {
        return env.get(s);
    }
}
```

Visitor pattern

```
public interface Visitor<R,A> {

    public R visit(PlusExp n, A arg);
    public R visit(MinusExp n, A arg);
    public R visit(TimesExp n, A arg);
    public R visit(Identifier n, A arg);
    public R visit(IntegerLiteral n, A arg);

}

public class Interpreter implements Visitor<Integer,Map<String,Integer>>{

    public Integer visit(PlusExp n,Map<String,Integer> env) {
        return n.e1.accept(this,env)+n.e2.accept(this,env);
    }
    public Integer visit(MinusExp n,Map<String,Integer> env) {
        return n.e1.accept(this,env)-n.e2.accept(this,env);
    }
    public Integer visit(TimesExp n,Map<String,Integer> env) {
        return n.e1.accept(this,env)*n.e2.accept(this,env);
    }
    public Integer visit(Identifier n,Map<String,Integer> env) {
        return env.get(n.s);
    }
    public Integer visit(IntegerLiteral n,Map<String,Integer> env) {
        return n.val;
    }

}
```

```
public abstract class Exp {
    public abstract <R,A> R accept(Visitor<R,A> v, A arg);
}
public class PlusExp extends Exp {
    public Exp e1,e2;
    public PlusExp(Exp a1, Exp a2) { e1=a1; e2=a2; }
    public <R,A> R accept(Visitor<R,A> v, A arg) {
        return v.visit(this, arg);
    };
}
public class MinusExp extends Exp {
    public Exp e1,e2;
    public MinusExp(Exp a1, Exp a2) { e1=a1; e2=a2; }
    public <R,A> R accept(Visitor<R,A> v, A arg) {
        return v.visit(this, arg);
    };
}
public class TimesExp extends Exp {
    public Exp e1,e2;
    public TimesExp(Exp a1, Exp a2) { e1=a1; e2=a2; }
    public <R,A> R accept(Visitor<R,A> v, A arg) {
        return v.visit(this, arg);
    };
}
public class IntegerLiteral extends Exp {
    public int val;
    public IntegerLiteral(int i) { val = i; }
    public <R,A> R accept(Visitor<R,A> v, A arg) {
        return v.visit(this, arg);
    };
}
public class Identifier extends Exp {
    public String s;
    public Identifier(String n) { s = n; }
    public <R,A> R accept(Visitor<R,A> v, A arg) {
        return v.visit(this, arg);
    };
}
```

Exercice

- Implémenter un visiteur pour convertir une expression arithmétique en chaîne de caractères.

Travail à réaliser dans le TP1

```
public class TP1RtlFlowGraph extends FlowGraph {

    public Node entry() {}

    public Object instr(Node n) {}

    public TP1RtlFlowGraph(Function f) {}

    public Set<Ident> def(Node node) {}

    public Set<Ident> use(Node node) {}

}
```

renvoie le noeud d'entrée

renvoie un objet de classe `Instr` ou `EndInstr`, ou la valeur `null` (noeud d'entrée)

Rendu

- Déposer sous Moodle votre fichier `TPX*.java`
- Au préalable, lancer dans le repertoire parent de `tpX`:

```
javac tpX/*.java
chmod u+x tpX.sh
./tpX.sh rtl/examples/Factorial.rtl
```

- Au besoin, ajouter des fichiers de tests .rtl en les éditant manuellement (les tester avec `rtl.sh`

Pour la prochaine séance

- Lire le chapitre 4.3 sur le visitor pattern.