# Introduction à la Sémantique

Luigi Santocanale

Laboratoire d'Informatique Fondamentale,
Centre de Mathématiques et Informatique,
39, rue Joliot-Curie - F-13453 Marseille

# Plan

1 Exemples

# Introduction à la Sémantique

Luigi Santocanale

Laboratoire d'Informatique Fondamentale,
Centre de Mathématiques et Informatique,
39, rue Joliot-Curie - F-13453 Marseille

# Plan

1 Exemples

# Premier exemple (langage C)

```
x = STARTX; y = STARTY;   z=0;
if(x) {
   y+=2;
   while(x){
      for(;y;y--) z++;
      x--;
      y+=2;
   }
}
```

Valeur de z à la ligne 18 ?

# Solution

Valeur de z :

```
STARTY + 2*STARTX        (STARTX > 0, STARTY >= 0)
```

Le code est *equivalent* à :

```
x = STARTX; y = STARTY;  z=0;
for(x = STARTX;x != 0;x--)
  for(y+=2;y != 0;y--) z++;
y+=2;
```

# Solution

Valeur de z :

```
STARTY + 2*STARTX        (STARTX > 0, STARTY >= 0)
```

Le code est *equivalent* à :

```
x = STARTX; y = STARTY;  z=0;
for(x = STARTX;x != 0;x--)
  for(y+=2;y != 0;y--) z++;
y+=2;
```

# Transformations – I

Test sur la condition booléenne :

```
if( x )   ->   if( x != 0 )
```

```
x = STARTX; y = STARTY;  z=0;
if(x != 0) {
  y+=2;
  while(x != 0){
    for(;y != 0;y--) z++;
    x--;
    y+=2;
  }
}
```

# Transformations – I

Test sur la condition booléenne :

```
if( x )   ->   if( x != 0 )
```

```
x = STARTX; y = STARTY;   z=0;
if(x !=  0) {
   y+=2;
   while(x != 0){
      for(;y != 0;y--) z++;
      x--;
      y+=2;
   }
}
```

# Transformations – II

« Rolling » :

```
c1;                          while(cond){
while(cond){        ->          c1;c2;
  c2;c1;                     }
}                            c1;
```

# Transformations – II

« Rolling » :

```
c1;                                  while(cond){
while(cond){          ->               c1;c2;
  c2;c1;                             }
}                                    c1;
```

```
x = STARTX; y = STARTY;  z=0;
if(x != 0) {
  y+=2;
  while(x != 0){
    for(;y != 0;y--) z++;
    x--;
    y+=2;
  }
}
```

# Transformations – II

« Rolling » :

```
c1;                                while(cond){
while(cond){          ->             c1;c2;
  c2;c1;                           }
}                                  c1;
```

```
x = STARTX; y = STARTY;  z=0;
if(x != 0) {
  while(x != 0){
    y+=2;
    for(;y != 0;y--) z++;
    x--;
  }
  y+=2;
}
```

# Transformations – III

Initialisation boucle `for` :

```
x = STARTX; y = STARTY;  z=0;
if(x != 0) {
   while(x != 0){
      y+=2;
      for(;y != 0;y--) z++;
      x--;
   }
   y+=2;
}
```

# Transformations – III

Initialisation boucle `for` :

```
x = STARTX; y = STARTY;   z=0;
if(x != 0) {
   while(x != 0){
      for(y+=2;y != 0;y--) z++;
      x--;
   }
   y+=2;
}
```

# Transformations – IV

Simplification `while` :

```
if(condition)                          while(condition)
  while(condition)          ->             command;
    command;
```

# Transformations – IV

Simplification `while` :

```
if(condition)                          while(condition)
  while(condition)          ->             command;
    command;
```

```
x = STARTX; y = STARTY;   z=0;
if(x != 0) {
  while(x != 0){
    for(y+=2;y != 0;y--) z++;
    x--;
  }
  y+=2;
}
```

# Transformations – IV

Simplification `while` :

```
if(condition)                          while(condition)
  while(condition)          ->             command;
    command;
```

```
x = STARTX; y = STARTY;  z=0;
while(x != 0){
  for(y+=2;y != 0;y--) z++;
  x--;
}
y+=2;
```

# Transformations – V

Simplification `while -> for` :

# Transformations – V

Simplification `while -> for` :

```
x = STARTX; y = STARTY;  z=0;
while(x != 0){
   for(y+=2;y != 0;y--) z++;
   x--;
}
y+=2;
```

# Transformations – V

Simplification `while -> for` :

```
x = STARTX; y = STARTY;  z=0;
for(;x != 0;x--)
  for(y+=2;y != 0;y--) z++;
y+=2;
```

# Transformations – V

Simplification `while -> for` :

```
x = STARTX; y = STARTY;  z=0;
for(x = STARTX;x != 0;x--)
  for(y+=2;y != 0;y--) z++;
y+=2;
```

# Deuxième exemple (langages fonctionnels)

Transformation code fonctionnel en
forme récursive terminale.

```
let rec length = function
    [] -> 0
  | testa::coda -> 1 + length coda
;;

let length l =
  let rec
    length_acc lista acc = match lista with
        [] -> acc
      | testa::coda -> length_acc coda (acc + 1)
  in
    length_acc l 0
;;
```

# Deuxième exemple (langages fonctionnels)

Transformation code fonctionnel en
forme récursive terminale.

```
let rec length = function
    [] -> 0
  | testa::coda -> 1 + length coda
;;

let length l =
  let rec
    length_acc lista acc = match lista with
        [] -> acc
      | testa::coda -> length_acc coda (acc + 1)
  in
    length_acc l 0
;;
```

# Deuxième exemple (langages fonctionnels)

Transformation code fonctionnel en
forme récursive terminale.

```
let rec length = function
    [] -> 0
  | testa::coda -> 1 + length coda
;;

let length l =
  let rec
    length_acc lista acc = match lista with
        [] -> acc
      | testa::coda -> length_acc coda (acc + 1)
  in
    length_acc l 0
;;
```

# Qui intervient dans le processus de transformation ?

- Programmeur, dont les objectifs sont
  rendre le code lisible, efficace.
- Compilateur, optimise le code.
- Définition du langage de programmation.
  Établi les transformation acceptables.

# Qui intervient dans le processus de transformation ?

- Programmeur, dont les objectifs sont
  rendre le code lisible, efficace.
- Compilateur, optimise le code.
- Définition du langage de programmation.
  Établi les transformation acceptables.

# Qui intervient dans le processus de transformation ?

- Programmeur, dont les objectifs sont
  rendre le code lisible, efficace.
- Compilateur, optimise le code.
- Définition du langage de programmation.
  Établi les transformation acceptables.