

Stage INFO5

Année scolaire 2023-2024

**Utilisation du typage graduel pour
améliorer la sécurité des logiciels**

Sommaire

- I - Introduction
 - Typage statique et dynamique
 - Typage Graduel
 - Objectif
- II - Lambda-Calcul
 - Lambda-calcul non typé
 - Lambda-calcul typé
 - Hindley-Milner
 - Lien avec Python
- III - Application
 - Architecture de PyRight
 - Placement des ajouts
 - Algorithme et implémentation
- IV - Demonstration
- V - Conclusion

Typage Statique et Dynamique

Statique

Dynamique

Typage Statique et Dynamique

Statique

- Vérification avant l'exécution
- Meilleure résistance aux bogues
- Développement plus long
- Demande plus de réflexion
- Limitations dues au typage

Dynamique

Statique

- Vérification avant l'exécution
- Meilleure résistance aux bogues
- Développement plus long
- Demande plus de réflexion
- Limitations dues au typage

```
public class Main {  
    public static void main(String[] args) {  
        int nombre = 10;  
        String message = "Bonjour";  
    }  
}
```

Dynamique

Statique

- Vérification avant l'exécution
- Meilleure résistance aux bogues
- Développement plus long
- Demande plus de réflexion
- Limitations dues au typage

```
public class Main {  
    public static void main(String[] args) {  
        int nombre = 10;  
        String message = "Bonjour";  
    }  
}
```

Dynamique

- Vérification pendant l'exécution
- Peut contenir des bogues à l'exécution
- Développement plus rapide
- Facilite l'écriture du code
- Permet plus de flexibilité

Typage Statique et Dynamique

Statique

- Vérification avant l'exécution
- Meilleure résistance aux bogues
- Développement plus long
- Demande plus de réflexion
- Limitations dues au typage

```
public class Main {  
    public static void main(String[] args) {  
        int nombre = 10;  
        String message = "Bonjour";  
    }  
}
```

Dynamique

- Vérification pendant l'exécution
- Peut contenir des bogues à l'exécution
- Développement plus rapide
- Facilite l'écriture du code
- Permet plus de flexibilité

```
def addition(list):  
    return list[0] + list[1]  
  
print(addition([5, 10]))  
print(addition(["Hello, ", "world!"]))  
print(addition([[1, 2], [3, 4]]))
```

Statique

Dynamique



Typage Graduel

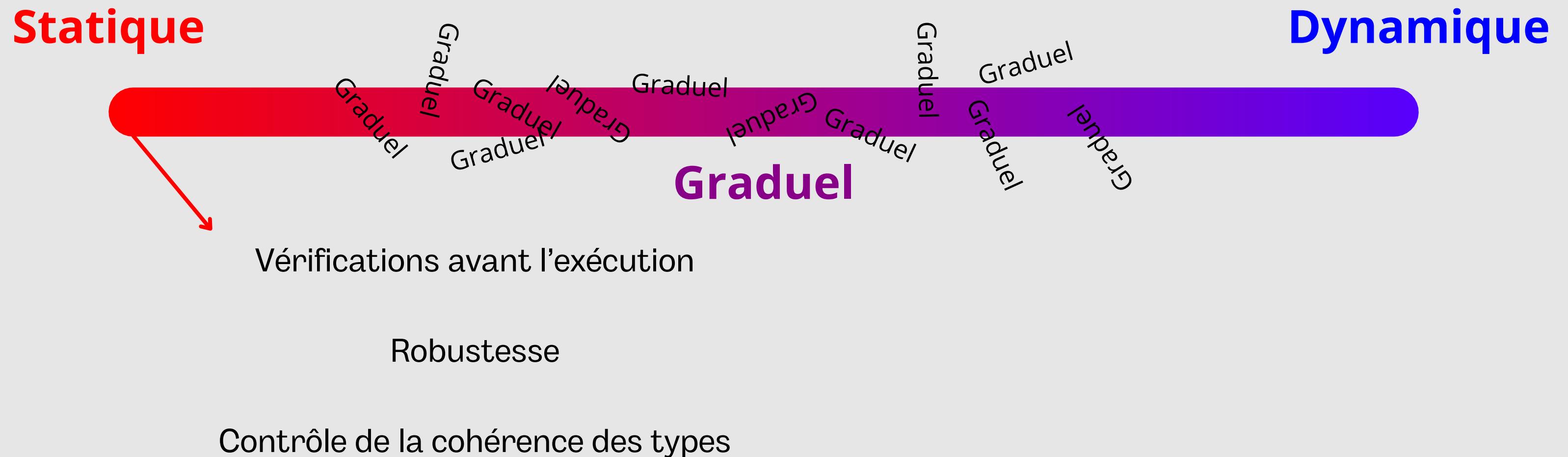
Statique

Dynamique

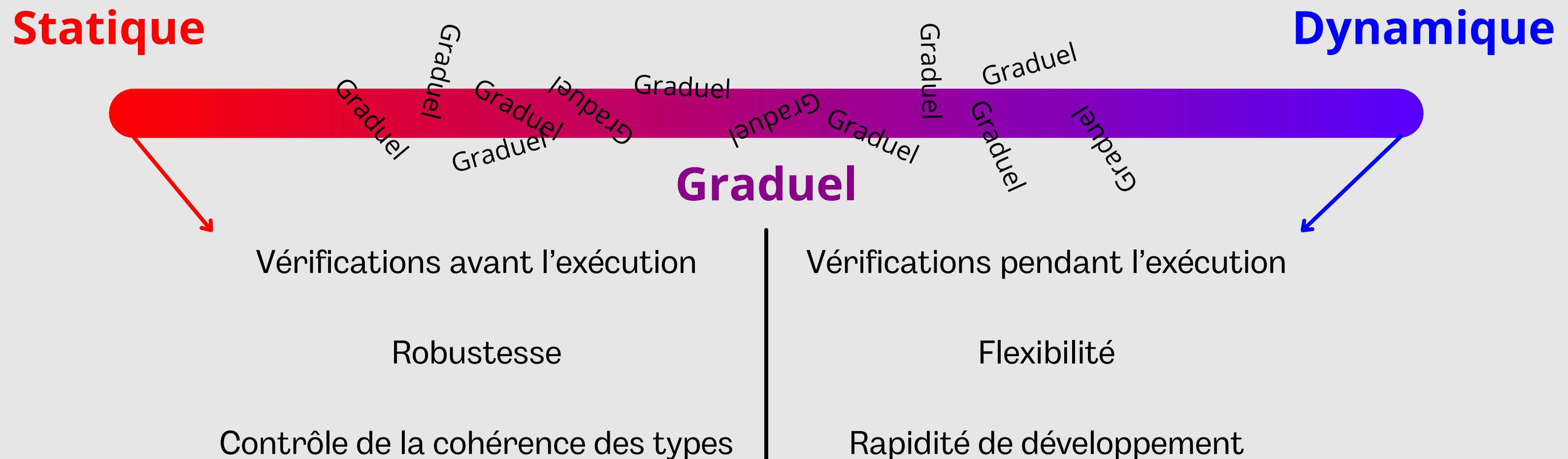
Graduel



Typage Graduel



Typage Graduel

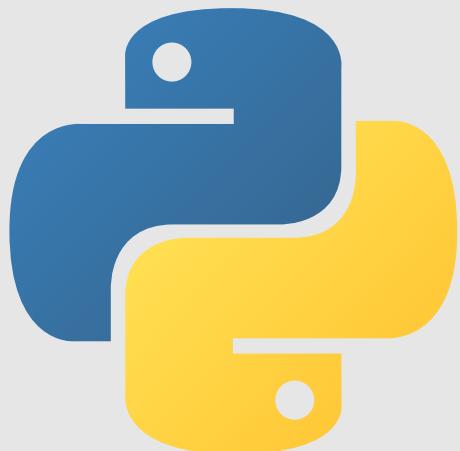


Appliquer à Python



Appliquer à Python

Améliorer la robustesse



Appliquer à Python

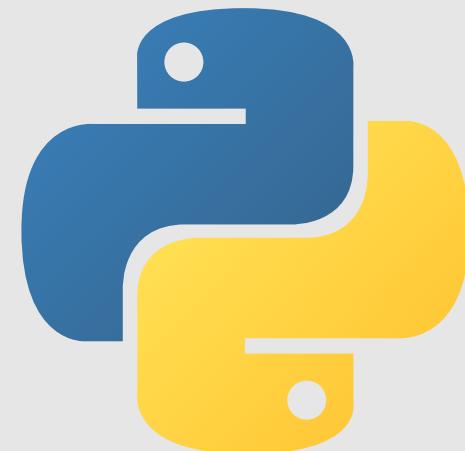
Améliorer la robustesse

Limiter l'intervention du développeur



Appliquer à Python

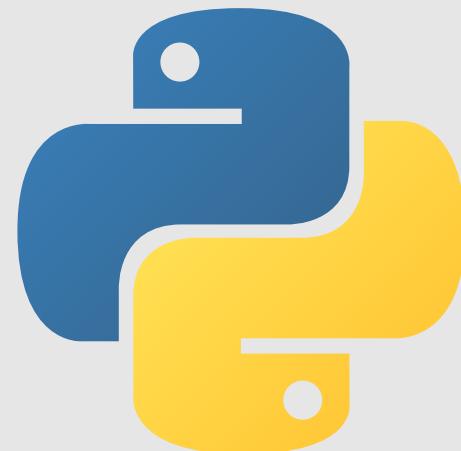
Améliorer la robustesse



Limiter l'intervention du développeur

Programme de typage déjà existant (PyRight)

Appliquer à Python



Améliorer la robustesse

Limiter l'intervention du développeur

Programme de typage déjà existant (PyRight)

Utilisation des indices de type

```
def f(p:int):
```

Améliorer PyRight



Améliorer PyRight

Gain de temps



Améliorer PyRight



Gain de temps

Support de Microsoft

Améliorer PyRight



Gain de temps

Support de Microsoft

Meilleurs résultats selon le site de référence
pour le typage en Python

<https://typing.readthedocs.io/>

Python Type System Conformance Test Results				
	mypy 1.11.1	pyright 1.1.374	pyre 0.9.22	pytype 2024.04.11
<u>Type annotations</u>				
annotations_coroutines	Pass	Pass	Pass	Pass
annotations_forward_refs	Partial	Pass	Partial	Partial
annotations_generators	Partial	Pass	Partial	Partial
annotations_methods	Pass*	Pass*	Pass*	Pass*
annotations_typeexpr	Pass	Pass	Pass	Partial
<u>Special types in annotations</u>				
specialtypes_any	Pass	Pass	Partial	Pass
specialtypes_never	Pass	Pass	Partial	Unsupported
specialtypes_none	Pass	Pass	Pass	Partial
specialtypes_promotions	Pass	Pass	Partial	Pass
specialtypes_type	Partial	Pass	Partial	Partial

Limitation de PyRight

```
def useInt(a:int):
    ...
    ...

def useString(a:str):
    ...
    ...

def foo(a):
    useInt(a)
    useString(a)
```

Limitation de PyRight

```
def useInt(a:int):
    ...
    ...

def useString(a:str):
    ...
    ...

def foo(a):
    useInt(a)
    useString(a)
```

Erreur de typage non détecté

Limitation de PyRight

Erreur détectable en utilisant l'algorithme d'inférence d'Hindley-Milner

```
def useInt(a:int):
    ...
def useString(a:str):
    ...
def foo(a):
    useInt(a)
    useString(a)
```

Erreur de typage non détecté

Syntaxe

Une lambda-expression e peut prendre 3 formes :

Syntaxe

Une lambda-expression e peut prendre 3 formes :

- Variables : x, y, z, \dots

Syntaxe

Une lambda-expression e peut prendre 3 formes :

- Variables : x, y, z, \dots
- Abstraction : $\lambda x \rightarrow e$

Syntaxe

Une lambda-expression e peut prendre 3 formes :

- Variables : x, y, z, \dots
- Abstraction : $\lambda x \rightarrow e$
- Application : $e_1 e_2$

Règles de calcul

Règles de calcul

- α - réduction : Remplacement d'un nom de variable par un autre.

$$(\lambda x \rightarrow x)_{[x \leftarrow z]} \Rightarrow \lambda z \rightarrow z$$

Règles de calcul

- α - réduction : Remplacement d'un nom de variable par un autre.

$$(\lambda x \rightarrow x)_{[x \leftarrow z]} \Rightarrow \lambda z \rightarrow z$$

- β - réduction : Application d'une lambda-expression à une abstraction.

$$\begin{aligned} (\lambda x \rightarrow e_1)e_2 &\xrightarrow{\beta} e_1[x \leftarrow e_2] \\ (\lambda x \rightarrow x\ y\ z)e &\xrightarrow{\beta} e\ y\ z \end{aligned}$$

Règles de calcul

- α - réduction : Remplacement d'un nom de variable par un autre.

$$(\lambda x \rightarrow x)_{[x \leftarrow z]} \Rightarrow \lambda z \rightarrow z$$

- β - réduction : Application d'une lambda-expression à une abstraction.

$$\begin{aligned} (\lambda x \rightarrow e_1)e_2 &\xrightarrow{\beta} e_1_{[x \leftarrow e_2]} \\ (\lambda x \rightarrow x\ y\ z)e &\xrightarrow{\beta} e\ y\ z \end{aligned}$$

- Priorité : Utilisation de l'expression la plus à gauche.

$$e_1\ e_2\ e_3 \Rightarrow ((e_1\ e_2)e_3)$$

Objectif : Reproduire le comportement des booléens

Objectif : Reproduire le comportement des booléens

2 critères :

$$\text{if } \text{true } e_1 \ e_2 \rightarrow e_1$$
$$\text{if } \text{false } e_1 \ e_2 \rightarrow e_2$$

Objectif : Reproduire le comportement des booléens

2 critères :

$$\text{if } \text{true } e_1 \ e_2 \rightarrow e_1$$

$$\text{if } \text{false } e_1 \ e_2 \rightarrow e_2$$

$$\text{if } = \lambda b \rightarrow (\lambda e_1 \rightarrow (\lambda e_2 \rightarrow (?)))$$

Objectif : Reproduire le comportement des booléens

2 critères :

$$if\ true\ e_1\ e_2 \rightarrow e_1$$

$$if\ false\ e_1\ e_2 \rightarrow e_2$$

$$if = \lambda b \rightarrow (\lambda e_1 \rightarrow (\lambda e_2 \rightarrow (?)))$$

$$true = \lambda x \rightarrow (\lambda y \rightarrow x)$$

$$false = \lambda x \rightarrow (\lambda y \rightarrow y)$$

$$if = \lambda b \rightarrow (\lambda e_1 \rightarrow (\lambda e_2 \rightarrow (b\ e_1\ e_2)))$$

Règles de calcul

Règles de calcul

Une lambda-expression typée e peut prendre 3 formes :

Règles de calcul

Une lambda-expression typée e peut prendre 3 formes :

- Variables : x, y, z, \dots (Pareil)

Règles de calcul

Une lambda-expression typée e peut prendre 3 formes :

- Variables : x, y, z, \dots (Pareil)
- Abstraction : $\lambda x : T \rightarrow e$ (Avec T un type)

Règles de calcul

Une lambda-expression typée e peut prendre 3 formes :

- Variables : x, y, z, \dots (Pareil)
- Abstraction : $\lambda x : T \rightarrow e$ (Avec T un type)
- Application : $e_1 e_2$ (Pareil)

Règles de calcul

Une lambda-expression typée e peut prendre 3 formes :

- Variables : x, y, z, \dots (Pareil)
- Abstraction : $\lambda x : T \rightarrow e$ (Avec T un type)
- Application : $e_1 e_2$ (Pareil)

Ajout de la notion de contexte Γ :

Règles de calcul

Une lambda-expression typée e peut prendre 3 formes :

- Variables : x, y, z, \dots (Pareil)
- Abstraction : $\lambda x : T \rightarrow e$ (Avec T un type)
- Application : $e_1 e_2$ (Pareil)

Ajout de la notion de contexte Γ :

- Γ est une liste de couples $(x : T)$ associant un type à un nom de variable

Syntaxe

Une lambda-expression typée e peut prendre 3 formes :

- Variables : x, y, z, \dots (Pareil)
- Abstraction : $\lambda x : T \rightarrow e$ (Avec T un type)
- Application : $e_1 e_2$ (Pareil)

Ajout de la notion de contexte Γ :

- Γ est une liste de couples $(x : T)$ associant un type à un nom de variable
- $\Gamma \vdash e : T$ signifie que l'expression e possède le type T dans le contexte Γ

Règles de Typage

Règles de Typage

- $\Gamma \vdash x : T$ si $x : T \in \Gamma$ $(x : \text{int}) \vdash x \Rightarrow x \text{ possède le type } \text{int}$

Règles de Typage

- $\Gamma \vdash x : T$ si $x : T \in \Gamma$ $(x : \text{int}) \vdash x \Rightarrow x$ possède le type *int*
- $\Gamma \vdash (\lambda x : T_1 \longrightarrow e) : (T_1 \rightarrow T_2)$ si $\Gamma \cup (x : T_1) \vdash e : T_2$ $(\lambda x : \text{int} \longrightarrow x + 1)$ possède le type *int* \rightarrow *int*

Règles de Typage

- $\Gamma \vdash x : T$ si $x : T \in \Gamma$ $(x : int) \vdash x \Rightarrow x$ possède le type int
- $\Gamma \vdash (\lambda x : T_1 \longrightarrow e) : (T_1 \rightarrow T_2)$ si $\Gamma \cup (x : T_1) \vdash e : T_2$ $(\lambda x : int \longrightarrow x + 1)$ possède le type $int \rightarrow int$
- $\Gamma \vdash e_1 e_2 : T'$ si $\Gamma \vdash e_1 : T \rightarrow T'$ et $\Gamma \vdash e_2 : T$ $(int \rightarrow int) \; int \Rightarrow int$

Règles de Typage

- $\Gamma \vdash x : T$ si $x : T \in \Gamma$ $(x : \text{int}) \vdash x \Rightarrow x$ possède le type *int*
- $\Gamma \vdash (\lambda x : T_1 \longrightarrow e) : (T_1 \rightarrow T_2)$ si $\Gamma \cup (x : T_1) \vdash e : T_2$ $(\lambda x : \text{int} \longrightarrow x + 1)$ possède le type *int* \rightarrow *int*
- $\Gamma \vdash e_1 e_2 : T'$ si $\Gamma \vdash e_1 : T \rightarrow T'$ et $\Gamma \vdash e_2 : T$ $(\text{int} \rightarrow \text{int}) \text{ int} \Rightarrow \text{int}$

Exemples de type

- Fonction de A vers B : $A \rightarrow B$

Règles de Typage

- $\Gamma \vdash x : T$ si $x : T \in \Gamma$ $(x : \text{int}) \vdash x \Rightarrow x$ possède le type *int*
- $\Gamma \vdash (\lambda x : T_1 \longrightarrow e) : (T_1 \rightarrow T_2)$ si $\Gamma \cup (x : T_1) \vdash e : T_2$ $(\lambda x : \text{int} \longrightarrow x + 1)$ possède le type *int* \rightarrow *int*
- $\Gamma \vdash e_1 e_2 : T'$ si $\Gamma \vdash e_1 : T \rightarrow T'$ et $\Gamma \vdash e_2 : T$ $(\text{int} \rightarrow \text{int}) \text{ int} \Rightarrow \text{int}$

Exemples de type

- Fonction de A vers B : $A \rightarrow B$
- Fonction utilisant une autre fonction : $(A \rightarrow B) \rightarrow A \rightarrow B$

Règles de Typage

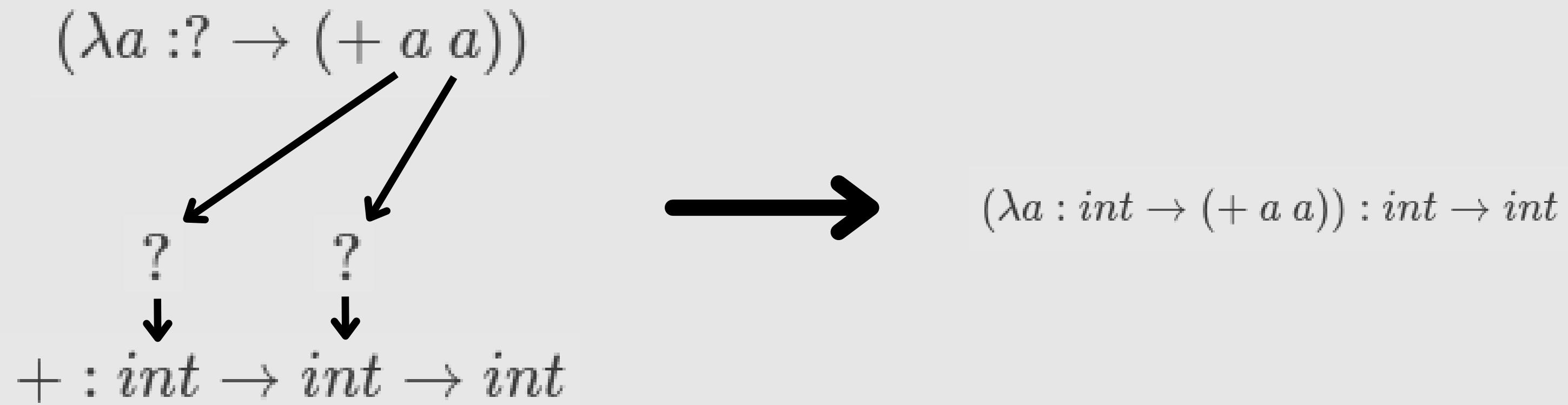
- $\Gamma \vdash x : T$ si $x : T \in \Gamma$ $(x : \text{int}) \vdash x \Rightarrow x \text{ possède le type } \text{int}$
- $\Gamma \vdash (\lambda x : T_1 \rightarrow e) : (T_1 \rightarrow T_2)$ si $\Gamma \cup (x : T_1) \vdash e : T_2$ $(\lambda x : \text{int} \rightarrow x + 1) \text{ possède le type } \text{int} \rightarrow \text{int}$
- $\Gamma \vdash e_1 e_2 : T'$ si $\Gamma \vdash e_1 : T \rightarrow T'$ et $\Gamma \vdash e_2 : T$ $(\text{int} \rightarrow \text{int}) \text{ int} \Rightarrow \text{int}$

Exemples de type

- Fonction de A vers B : $A \rightarrow B$
- Fonction utilisant une autre fonction : $(A \rightarrow B) \rightarrow A \rightarrow B$
- Abomination : $(A \rightarrow (D \rightarrow B) \rightarrow C \rightarrow B) \rightarrow ((A \rightarrow (D \rightarrow B) \rightarrow C \rightarrow B) \rightarrow A \rightarrow (A \rightarrow (D \rightarrow B) \rightarrow C \rightarrow B) \rightarrow A)$

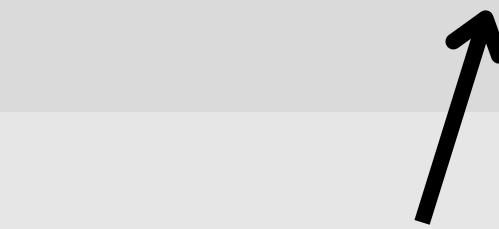
Inférer un type

Inférer un type



$f : ? \rightarrow ?$

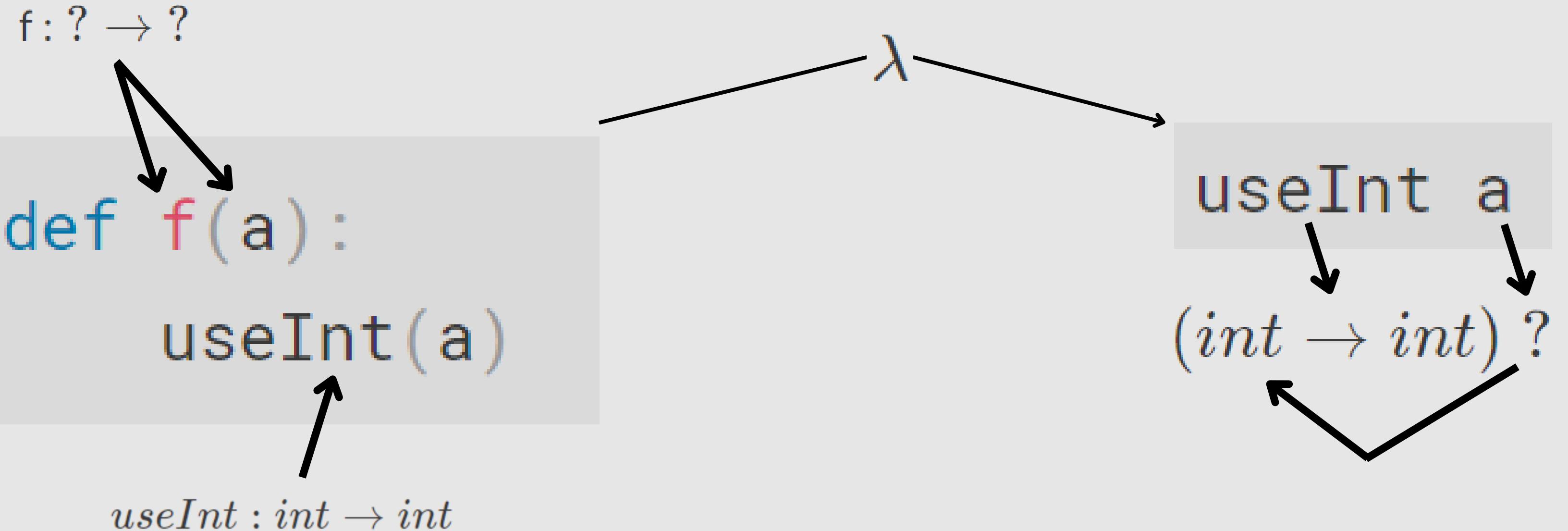
```
def f(a):  
    useInt(a)
```

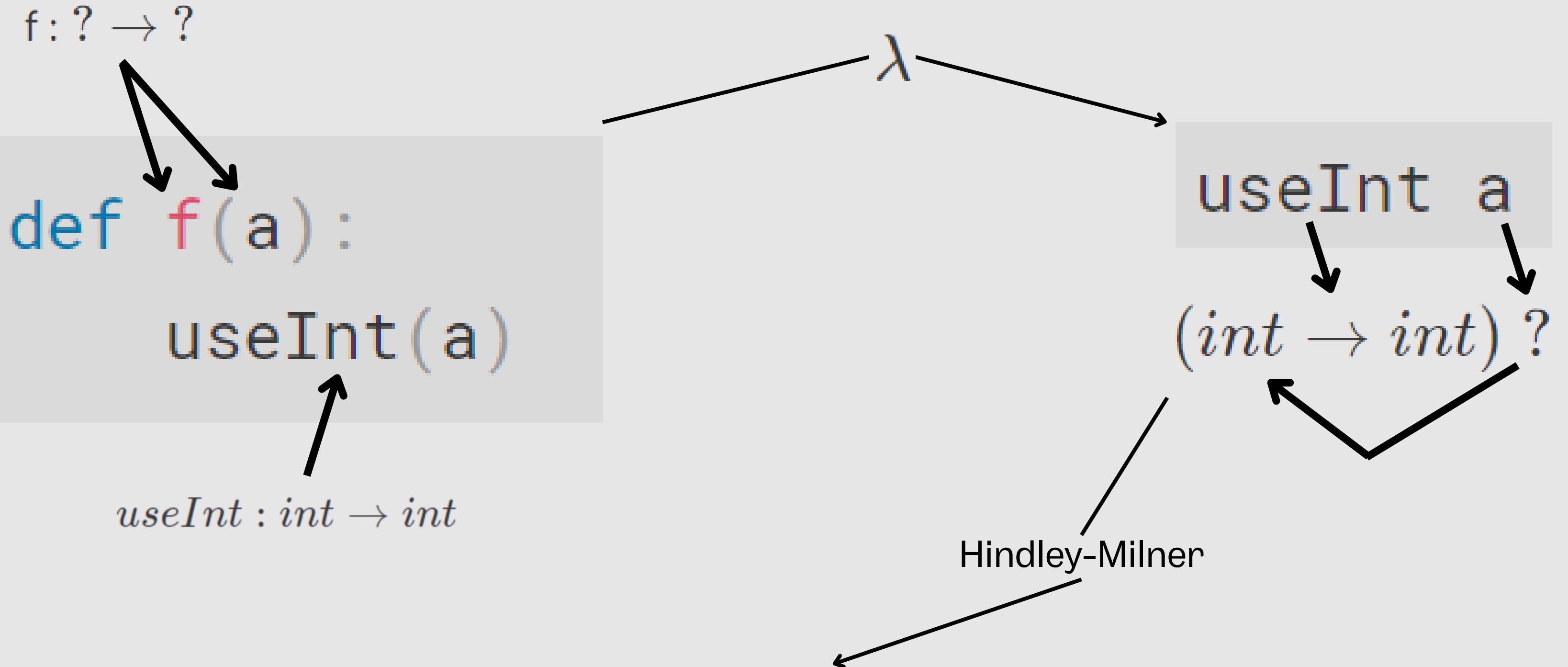
 $useInt : int \rightarrow int$

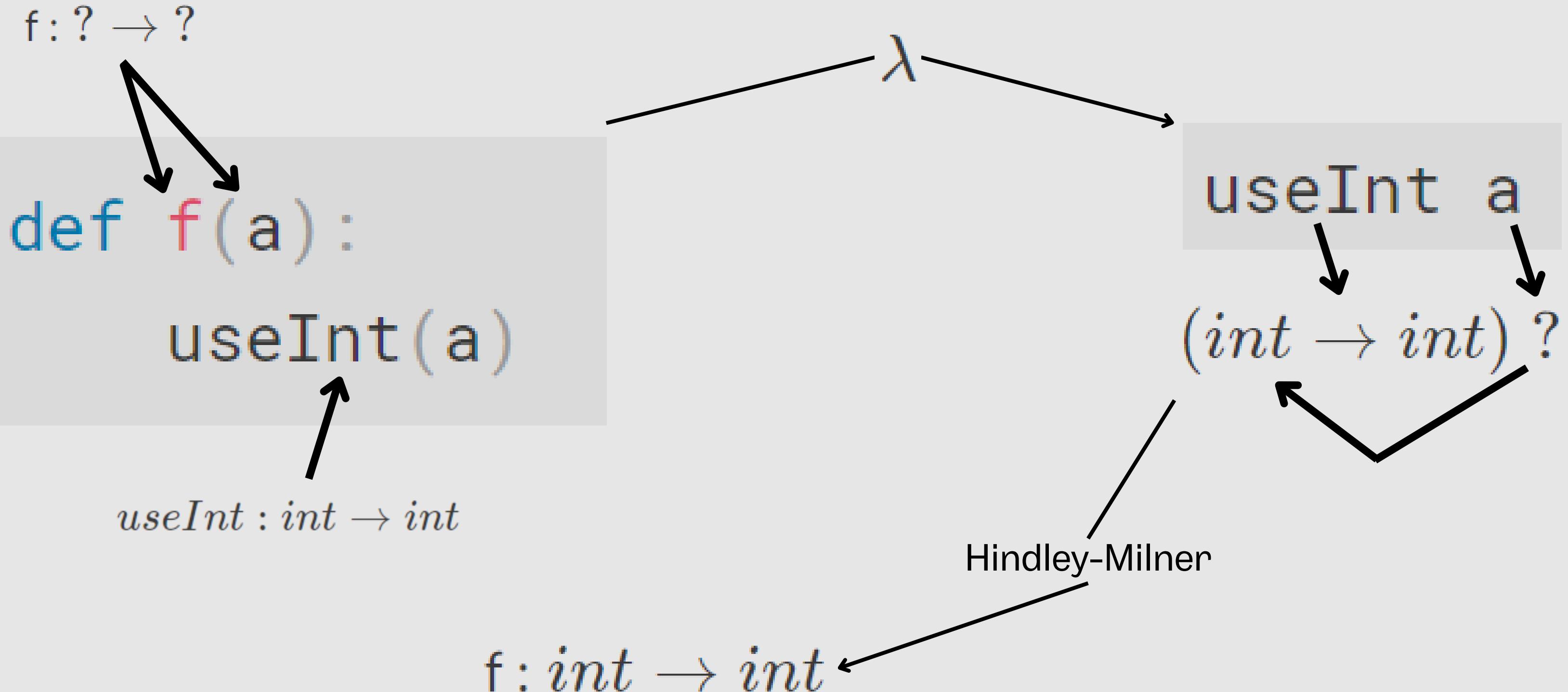
$f : ? \rightarrow ?$

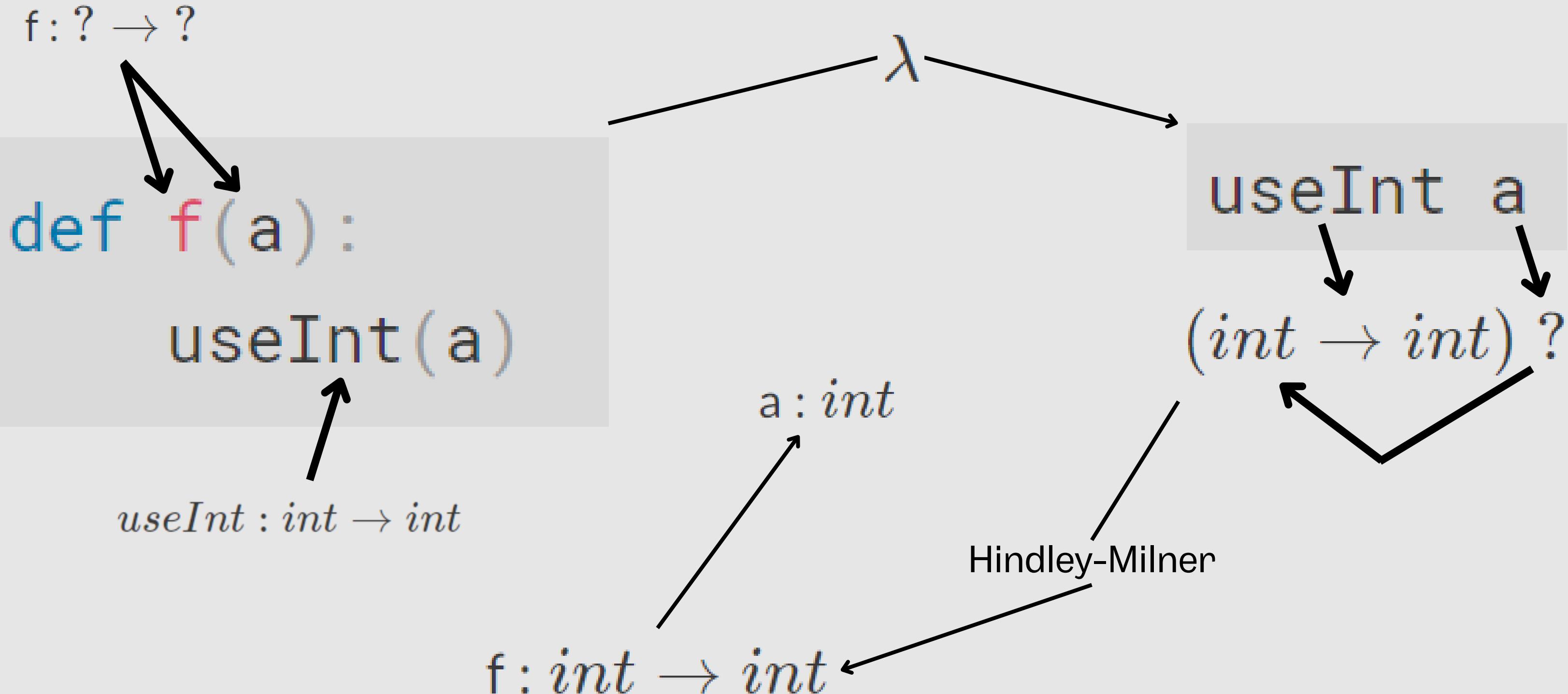
```
def f(a):  
    useInt(a)
```

 $useInt : int \rightarrow int$ λ

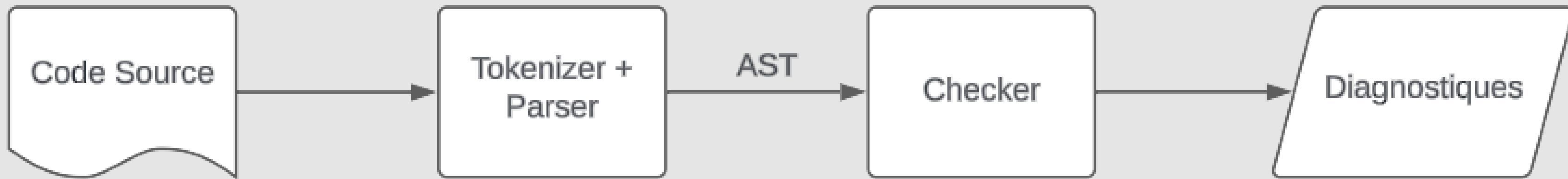






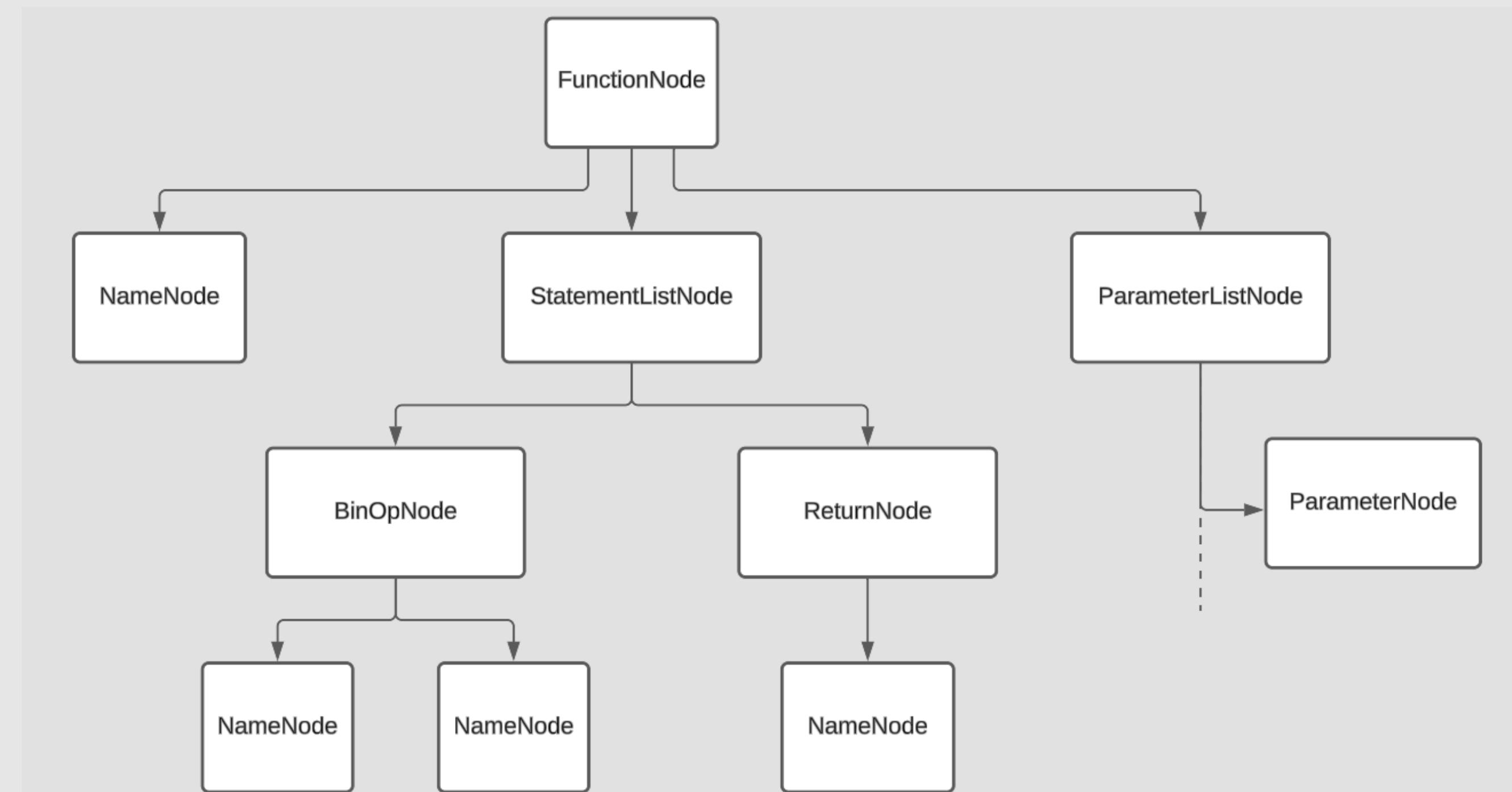


Processus de PyRight



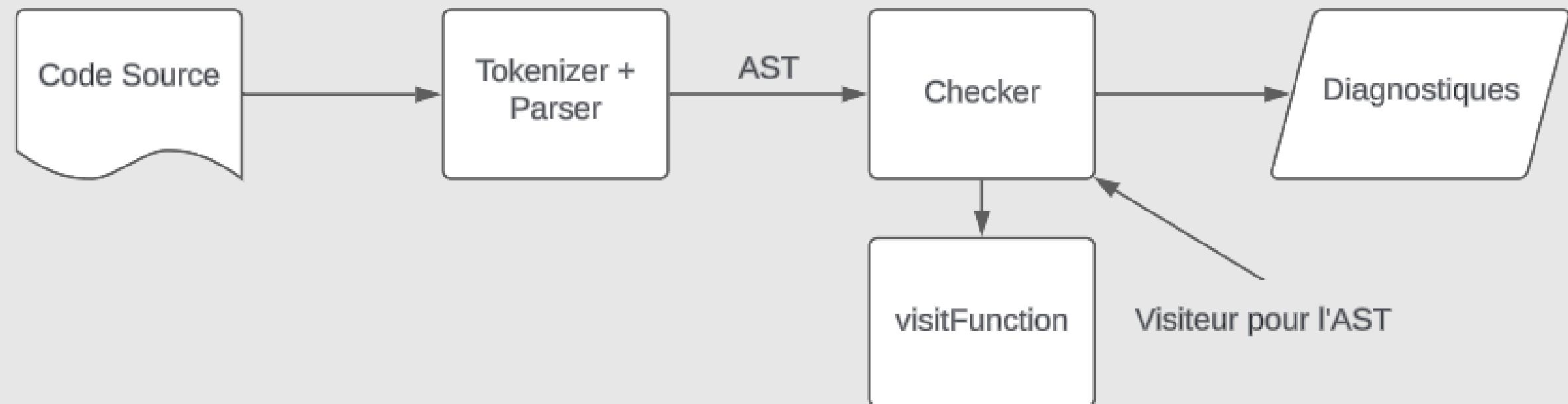
```
def add(a, b):  
    c = a+b  
    return c
```

Exemple d'AST



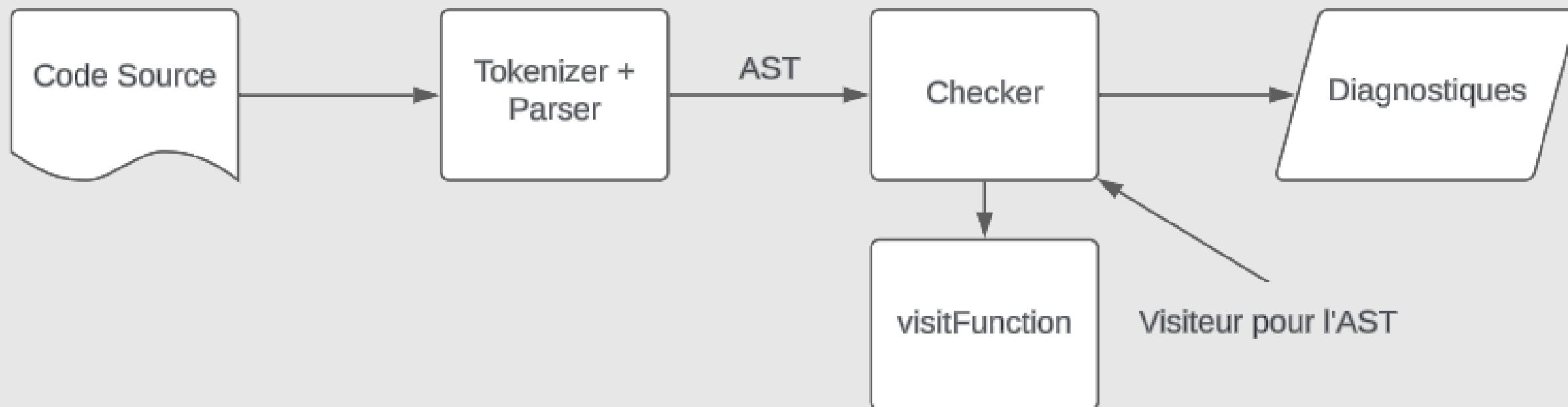
Placement des Ajouts

Placement des Ajouts



Placement des Ajouts

Placement des Ajouts



```
override visitClass(node: ClassNode): boolean { ... }  
override visitFunction(node: FunctionNode): boolean { ... }  
override visitLambda(node: LambdaNode): boolean { ... }  
override visitCall(node: CallNode): boolean { ... }
```

Inférer le type d'un paramètre

```
def func(a):
    useInt(a)
    useStr(a)
    return
```

```
function inferParameters(node: FunctionNode) {
    let callsites = getCallsites(node)
    for(parameter in node.parameters){
        let signatures = getSignatures(parameter, callsites)
        let candidates = getCandidates(parameter, signatures)
        let typeToInfer = intersection(candidates)
        if(typeToInfer.isEmpty()){
            addDiagnostic("Echec de l'inférence")
        } else if(typeToInfer.length == 1){
            setParameterType(typeToInfer[0])
        } else {
            setParameterType(createUnionType(typeToInfer))
        }
    }
}
```

Inférer le type d'un paramètre

```
def func(a):
    useInt(a)
    useStr(a)
    return
```

```
function inferParameters(node: FunctionNode) {
    {useInt, useStr} ← let callsites = getCallsites(node)
    for(parameter in node.parameters){
        let signatures = getSignatures(parameter, callsites)
        let candidates = getCandidates(parameter, signatures)
        let typeToInfer = intersection(candidates)
        if(typeToInfer.isEmpty()){
            addDiagnostic("Echec de l'inférence")
        } else if(typeToInfer.length == 1){
            setParameterType(typeToInfer[0])
        } else {
            setParameterType(createUnionType(typeToInfer))
        }
    }
}
```

Inférer le type d'un paramètre

```
def func(a):
    useInt(a)
    useStr(a)
    return
```

```
function inferParameters(node: FunctionNode) {
    {useInt, useStr} ← let callsites = getCallsites(node)
    {int → int, str → str} ← for(parameter in node.parameters){
        let signatures = getSignatures(parameter, callsites)
        let candidates = getCandidates(parameter, signatures)
        let typeToInfer = intersection(candidates)
        if(typeToInfer.isEmpty()){
            addDiagnostic("Echec de l'inférence")
        } else if(typeToInfer.length == 1){
            setParameterType(typeToInfer[0])
        } else {
            setParameterType(createUnionType(typeToInfer))
        }
    }
}
```

Inférer le type d'un paramètre

```
def func(a):
    useInt(a)
    useStr(a)
    return
```

```
function inferParameters(node: FunctionNode) {
    {useInt, useStr} ← let callsites = getCallsites(node)
    {int → int, str → str} ← for(parameter in node.parameters){
        let signatures = getSignatures(parameter, callsites)
        {int, str} ← let candidates = getCandidates(parameter, signatures)
        let typeToInfer = intersection(candidates)
        if(typeToInfer.isEmpty()){
            addDiagnostic("Echec de l'inférence")
        } else if(typeToInfer.length == 1){
            setParameterType(typeToInfer[0])
        } else {
            setParameterType(createUnionType(typeToInfer))
        }
    }
}
```

Inférer le type d'un paramètre

```
def func(a):
    useInt(a)
    useStr(a)
    return
```

```
function inferParameters(node: FunctionNode) {
    {useInt, useStr} ← let callsites = getCallsites(node)
    {int → int, str → str} ← for(parameter in node.parameters){
        let signatures = getSignatures(parameter, callsites)
        {int, str} ← let candidates = getCandidates(parameter, signatures)
        {} ← let typeToInfer = intersection(candidates)
        if(typeToInfer.isEmpty()){
            addDiagnostic("Echec de l'inférence")
        } else if(typeToInfer.length == 1){
            setParameterType(typeToInfer[0])
        } else {
            setParameterType(createUnionType(typeToInfer))
        }
    }
}
```

Environnement de développement

The screenshot displays two instances of Visual Studio Code side-by-side, illustrating a multi-language development environment.

Left Window (checkerExtension.ts - pyright - Visual Studio Code):

- Title Bar:** checkerExtension.ts - pyright - Visual Studio Code
- File Menu:** File Edit Selection View Go Run Terminal Help
- Toolbar:** RUN AND DEBUG, Pyright exte..., etc.
- Editor:** Shows TypeScript code for a checker extension. The code includes methods like `getSignatureFromCall`, `findIntersection`, and `getTypeCandidates`.
- Sidebar:**
 - WATCH:** Shows variables: `fileToCheck = not available`, `this._ fileInfo = not available`, `fileInfo = not available`.
 - CALL STACK:** Shows a single entry: "Pyright extension (watch): E... RUNNING".
 - LOADED SCRIPTS:** Shows breakpoints for multiple files, including `checkerExtension.ts` at line 222:27.
 - BREAKPOINTS:** Shows caught and uncaught exceptions, along with breakpoints for various files.
 - EVENT LISTENER BREAKPOINTS:** Shows event listener breakpoints for various files.
- Bottom:** PROBLEMS, OUTPUT, DEBUG CONSOLE (selected), TERMINAL, PORTS. DEBUG CONSOLE shows the message "Filter (e.g. text, !exclude, |escape)".

Right Window ([Extension Development Host] test_pyright.py - My...):

- Title Bar:** [Extension Development Host] test_pyright.py - My...
- File Menu:** File Edit Selection View Go Run Terminal Help
- Editor:** Shows Python test code named `test_pyright.py`.


```
def Int(n:int):
    return n

def Str(s:str):
    return s

def func(a):
    Int(a)
    Str(a)
    return a
```
- Bottom:** Shows a warning message: "[ms-pyright.vscode-pyright]: Cannot register 'python.venvPath'. This..."

Environnement de développement

The screenshot shows two instances of Visual Studio Code side-by-side. The left instance is titled 'checkerExtension.ts - pyright - Visual Studio Code' and contains TypeScript code for a checker extension. The right instance is titled '[Extension Development Host] test_pyright.py - My...' and contains Python test code. Both files are displayed with syntax highlighting and code completion suggestions.

```
checkerExtension.ts - pyright - Visual Studio Code
File Edit Selection View Go Run Terminal Help
RUN AND DEBUG Pyright exte... e.ts M ts checker.ts M ts typeEvaluator.ts ts checkerExtension.ts U ts parseTreeWalker.ts ts parseNodes.ts ts parser.ts ...
packages > pyright-internal > src > analyzer > ts checkerExtension.ts > MyChecker
29  export class MyChecker extends Checker {
395      getSignatureFromCall(callsite: CallNode): ParameterNode[] {
396          let inferenceInfo = this._evaluator.getTypeOfExpression(callsite).overloadsUsedForCall
397          if (inferenceInfo) {
398              return inferenceInfo[0].details.declaration?.node.parameters || []
399          }
400          return []
401      }
402
403      findIntersection(types: Type[]): Type | undefined {
404          if (types.length == 0) return undefined;
405          let intersection: Type | undefined = types[0]
406          types.forEach((t) => {
407              if (intersection === undefined) return;
408              intersection = findCommonType(intersection, t)
409          })
410          return intersection;
411      }
412
413      getTypeCandidates(name: NameNode, callsites: CallNode[]): Type[] {
414          let candidates: Type[] = []
415          let parameterCallsites = callsites.filter(callsite => isParameterInCallsite(name, callsite))
416
417          parameterCallsites.forEach(callsite => {
418              //recover the rank of the parameter in the callsite
419              let argRank = callsite.arguments.findIndex(arg =>
420                  arg.valueExpression.nodeType === ParseNodeType.Name
421                  && arg.valueExpression.value === name.value)
422
423              let signature = this.getSignatureFromCall(callsite)
424              let annotation = signature[argRank]?.typeAnnotation
425              if (annotation?.nodeType === ParseNodeType.Name) {
426                  candidates.push([annotation.value])
427              } else if (annotation?.nodeType === ParseNodeType.StringList) {
428                  candidates.push(annotation.strings[0].value.split("|"))
429              }
430
431          })
432
433          return candidates
434      }
}

```

```
[Extension Development Host] test_pyright.py - My...
File Edit Selection View Go Run Terminal Help
test_pyright.py 1 ...
1 def Int(n:int):
2     return n
3
4 def Str(s:str):
5     return s
6
7 def func(a):
8     Int(a)
9     Str(a)
10    return a
11
12
13
14
```

- Mettre en place l'environnement

Environnement de développement

The screenshot shows two side-by-side code editors in Visual Studio Code. The left editor is for 'checkerExtension.ts' and the right is for 'test_pyright.py'. Both files are part of an 'Extension Development Host'.

checkerExtension.ts - pyright - Visual Studio Code

```

checkerExtension.ts - pyright - Visual Studio Code
File Edit Selection View Go Run Terminal Help
RUN AND DEBUG Pyright exte... e.ts M ts checker.ts M ts typeEvaluator.ts ts checkerExtension.ts U ts parseTreeWalker.ts ts parseNodes.ts ts parser.ts ...
packages > pyright-internal > src > analyzer > ts checkerExtension.ts > MyChecker
29  export class MyChecker extends Checker {
395      getSignatureFromCall(callsite: CallNode): ParameterNode[] {
396          let inferenceInfo = this._evaluator.getTypeOfExpression(callsite).overloadsUsedForCall
397          if (inferenceInfo) {
398              return inferenceInfo[0].details.declaration?.node.parameters || []
399          }
400          return []
401      }
402
403      findIntersection(types: Type[]): Type | undefined {
404          if (types.length == 0) return undefined;
405          let intersection: Type | undefined = types[0]
406          types.forEach((t) => {
407              if (intersection === undefined) return;
408              intersection = findCommonType(intersection, t)
409          })
410          return intersection;
411      }
412
413      getTypeCandidates(name: NameNode, callsites: CallNode[]): Type[] {
414          let candidates: Type[] = []
415          let parameterCallsites = callsites.filter(callsite => isParameterInCallsite(name, callsite))
416
417          parameterCallsites.forEach(callsite => {
418              //recover the rank of the parameter in the callsite
419              let argRank = callsite.arguments.findIndex(arg =>
420                  arg.valueExpression.nodeType === ParseNodeType.Name
421                  && arg.valueExpression.value === name.value)
422
423              let signature = this.getSignatureFromCall(callsite)
424              let annotation = signature[argRank]?.typeAnnotation
425              if (annotation?.nodeType === ParseNodeType.Name) {
426                  candidates.push([annotation.value])
427              } else if (annotation?.nodeType === ParseNodeType.StringList) {
428                  candidates.push(annotation.strings[0].value.split("|"))
429              }
430
431          })
432
433          return candidates
434      }
    
```

[Extension Development Host] test_pyright.py - My...

```

test_pyright.py 1
def Int(n:int):
    return n

def Str(s:str):
    return s

def func(a):
    Int(a)
    Str(a)
    return a
    
```

- Mettre en place l'environnement
- Coder en TypeScript

Environnement de développement

The screenshot shows two instances of Visual Studio Code side-by-side. The left instance is titled 'checkerExtension.ts - pyright - Visual Studio Code' and contains TypeScript code for a checker extension. The right instance is titled '[Extension Development Host] test_pyright.py - My...' and contains Python test code. Both files are part of an 'Extension Development Host' project.

```

checkerExtension.ts - pyright - Visual Studio Code
File Edit Selection View Go Run Terminal Help
RUN AND DEBUG Pyright exte... e.ts M ts checker.ts M ts typeEvaluator.ts ts checkerExtension.ts U ts parseTreeWalker.ts ts parseNodes.ts ts parser.ts ...
packages > pyright-internal > src > analyzer > ts checkerExtension.ts > MyChecker
29 export class MyChecker extends Checker {
395     getSignatureFromCall(callsite: CallNode): ParameterNode[] {
396         let inferenceInfo = this._evaluator.getTypeOfExpression(callsite).overloadsUsedForCall
397         if (inferenceInfo) {
398             return inferenceInfo[0].details.declaration?.node.parameters || []
399         }
400         return []
401     }
402
403     findIntersection(types: Type[]): Type | undefined {
404         if (types.length == 0) return undefined;
405         let intersection: Type | undefined = types[0]
406         types.forEach((t) => {
407             if (intersection === undefined) return;
408             intersection = findCommonType(intersection, t)
409         })
410         return intersection;
411     }
412
413     getTypeCandidates(name: NameNode, callsites: CallNode[]): Type[] {
414         let candidates: Type[] = []
415         let parameterCallsites = callsites.filter(callsite => isParameterInCallsite(name, callsite))
416
417         parameterCallsites.forEach(callsite => {
418             //recover the rank of the parameter in the callsite
419             let argRank = callsite.arguments.findIndex(arg =>
420                 arg.valueExpression.nodeType === ParseNodeType.Name
421                 && arg.valueExpression.value === name.value)
422             let signature = this.getSignatureFromCall(callsite)
423             let annotation = signature[argRank]?.typeAnnotation
424             if (annotation?.nodeType === ParseNodeType.Name) {
425                 candidates.push(annotation.value)
426             } else if (annotation?.nodeType === ParseNodeType.StringList) {
427                 candidates.push(annotation.strings[0].value.split("|"))
428             }
429         })
430
431         return candidates
432     }
433
434 }

test_pyright.py 1
def Int(n:int):
    return n

def Str(s:str):
    return s

def func(a):
    Int(a)
    Str(a)
    return a

```

The left panel includes a 'WATCH' sidebar showing variables: fileToCheck = not available, this._fileInfo = not available, fileInfo = not available. It also has a 'CALL STACK' section showing 'Pyright extension (watch): E... RUNNING'. The right panel has a 'PROBLEMS' and 'OUTPUT' tab at the bottom.

- Mettre en place l'environnement
- Coder en TypeScript
- Comprendre l'architecture de PyRight

Inférence réussie

```
def useStr(s:"str"):  
    return s  
  
def func(a):  
    useStr(a)  
    return a
```

Inférence réussie

```
def useStr(s:"str"):  
    return s  
  
def func(a):  
    useStr(a)  
    return a
```

Inférence réussie

```
def useIntOrStr(n:"int|str"):  
    return n  
  
def useStr(s:"str"):  
    return s  
  
def func(a):  
    useIntOrStr(a)  
    useStr(a)  
    return a
```

Inférence réussie

```
def useStr(s:"str"):  
    return s  
  
def func(a):  
    useStr(a)  
    return a
```

Inférence réussie

```
def useIntOrStr(n:"int|str"):  
    return n  
  
def useStr(s:"str"):  
    return s  
  
def func(a):  
    useIntOrStr(a)  
    useStr(a)  
    return a
```

Inférence échouée

```
def useInt(n:"int"):  
    return n  
  
def useStr(s:"str"):  
    return s  
  
def func(a):  
    useInt(a)  
    useStr(a)  
    return a
```

V- Conclusion

V- Conclusion

- Détection de nouvelles erreurs de type

V- Conclusion

- Détection de nouvelles erreurs de type
- Améliorations possibles :

V- Conclusion

- Détection de nouvelles erreurs de type
- Améliorations possibles :
 - Ajouter de nouvelles manières de récupérer des candidats

V- Conclusion

- Détection de nouvelles erreurs de type
- Améliorations possibles :
 - Ajouter de nouvelles manières de récupérer des candidats
 - Ajouter de nouvelles manières de calculer le type à inférer

Merci pour votre écoute !

Objectif : Reproduire le comportement des nombres entiers

Alonzo Church



"Compter, c'est répéter une opération!"

(C'est pas vrai il a pas dis ça ... mais presque!)

1 critère :

Pouvoir compter!

$$\text{Zéro} = \lambda f \rightarrow (\lambda x \rightarrow (x))$$

$$\text{Un} = \lambda f \rightarrow (\lambda x \rightarrow (fx))$$

$$\text{Deux} = \lambda f \rightarrow (\lambda x \rightarrow (ffx))$$

$$+ = \lambda mnfx \rightarrow (mf(nfx))$$

+ Un Deux

$$\Rightarrow (\lambda mnfx \rightarrow (mf(nfx))) (\lambda fx \rightarrow (fx)) (\lambda fx \rightarrow (f(fx)))$$

$$\Rightarrow (\lambda nfx \rightarrow ((\lambda fx \rightarrow (fx))f(nfx))) (\lambda fx \rightarrow (f(fx)))$$

$$\Rightarrow (\lambda fx \rightarrow ((\lambda fx \rightarrow (fx))f((\lambda fx \rightarrow (f(fx)))fx)))$$

$$\Rightarrow (\lambda fx \rightarrow ((\lambda fx \rightarrow (fx))f(f(fx)))))$$

$$\Rightarrow (\lambda fx \rightarrow ((\lambda x \rightarrow (fx))(f(fx)))))$$

$$\Rightarrow \lambda fx \rightarrow (f(f(fx))))$$

\Rightarrow Trois